Distributed Systems

Chun-Feng Liao

廖峻鋒

Department of Computer Science

National Chengchi University

Distributed Systems

Networking

Chun-Feng Liao

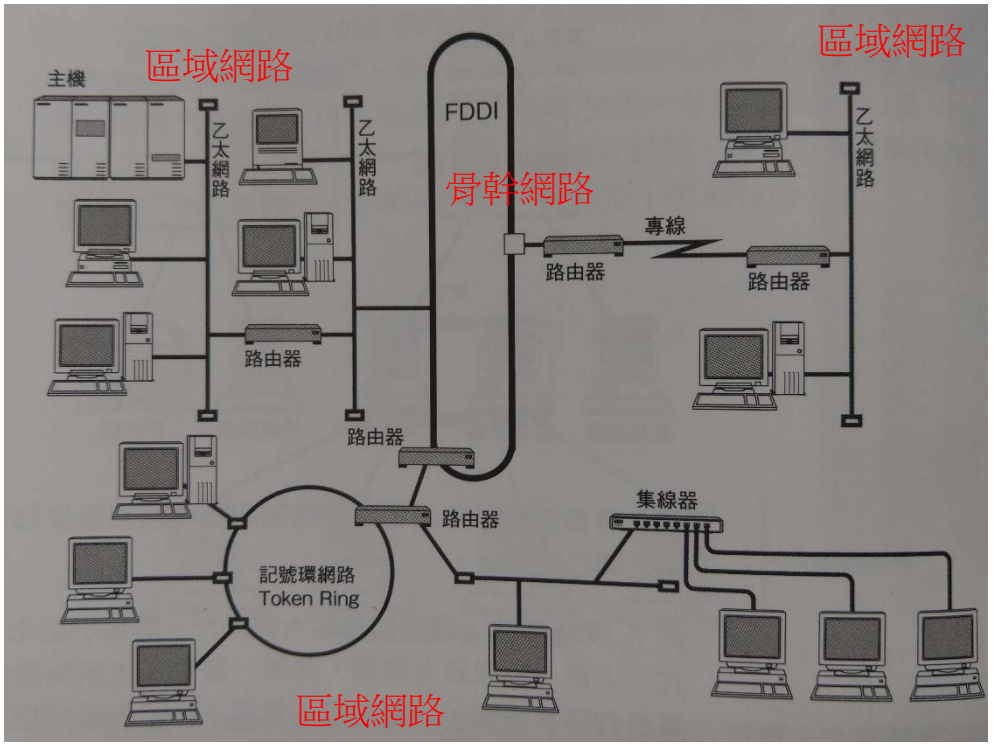
廖峻鋒

Dept. of Computer Science

National Chengchi University

Introduction

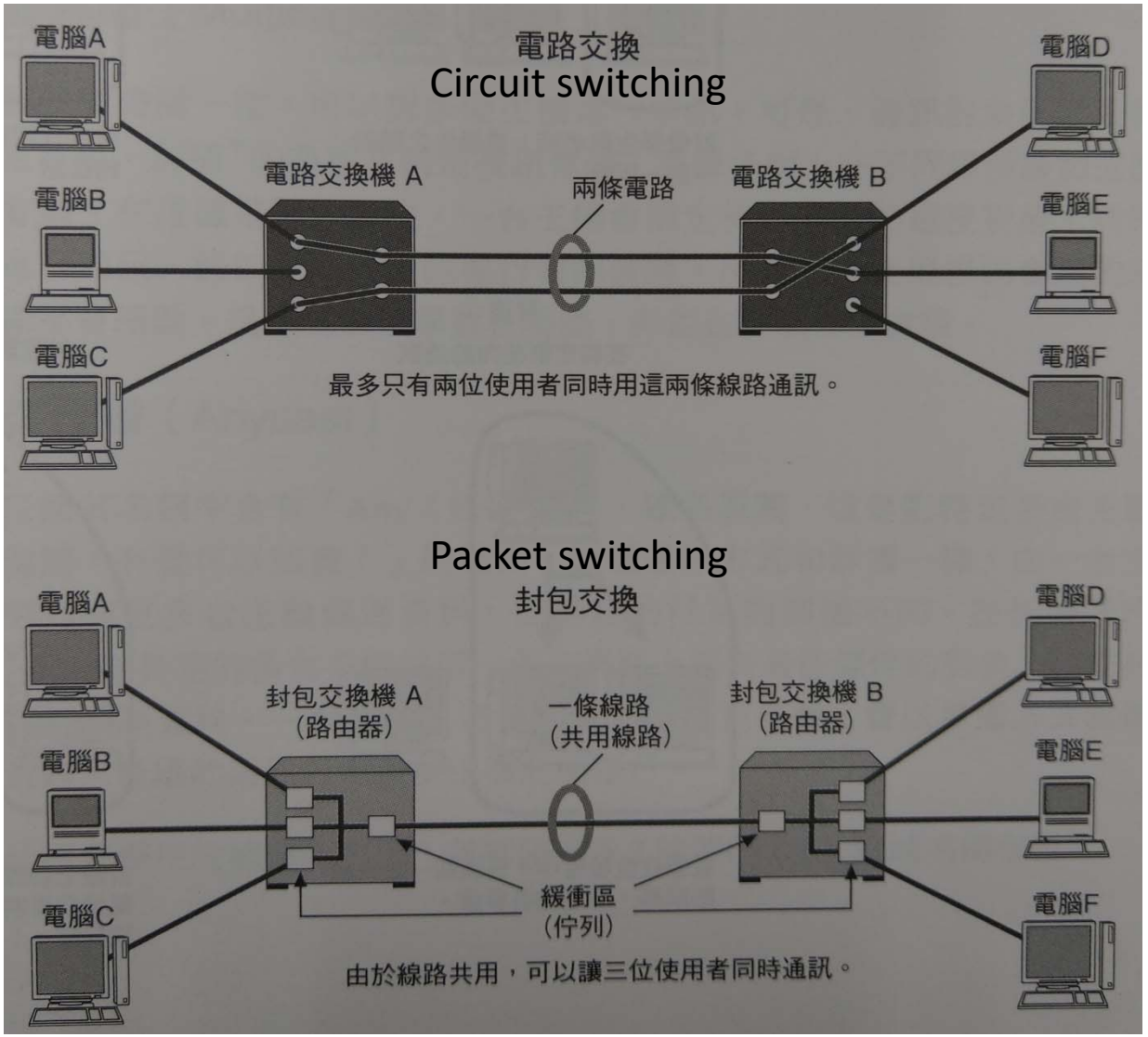
- Communication network of a distributed system
  - 實體節點 Device: router, switch, bridges, hubs, repeaters, 網卡
  - 節點間通訊連結 Connection: wire, cable, fiber, 無線電
  - 通訊控制 Software: protocol, drivers, API



Switching Network

- Switching
  - 判斷並執行「那個訊息往那裡送」的機制
- Circuit switching
  - Two nodes establish a dedicated communications channel (circuit)
  - 傳統電信網路(POTS, Plain old telephone system)
- Packet switching
  - Packet: basic unit of transmission over network
    - 繞送資訊會放在封包表頭中
  - Packets are queued in a buffer
    - transmitted when the link is available
    - transmitted over a shared network channel

Circuit switching vs. Packet switching

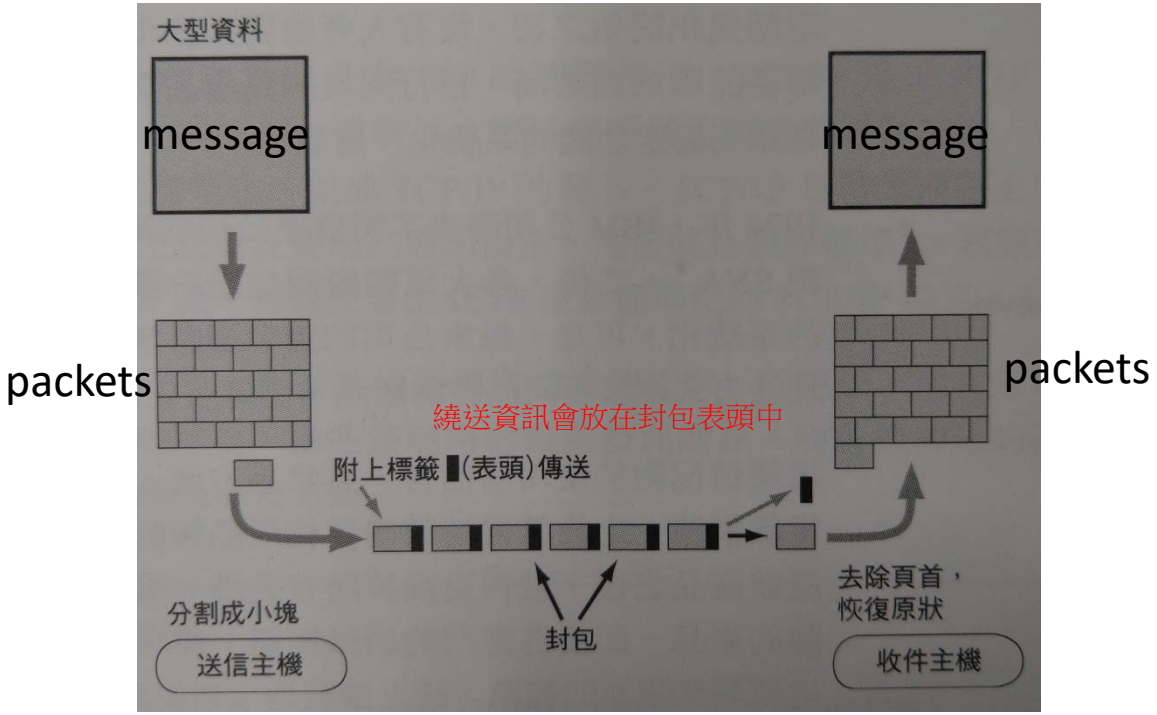


From message to packet

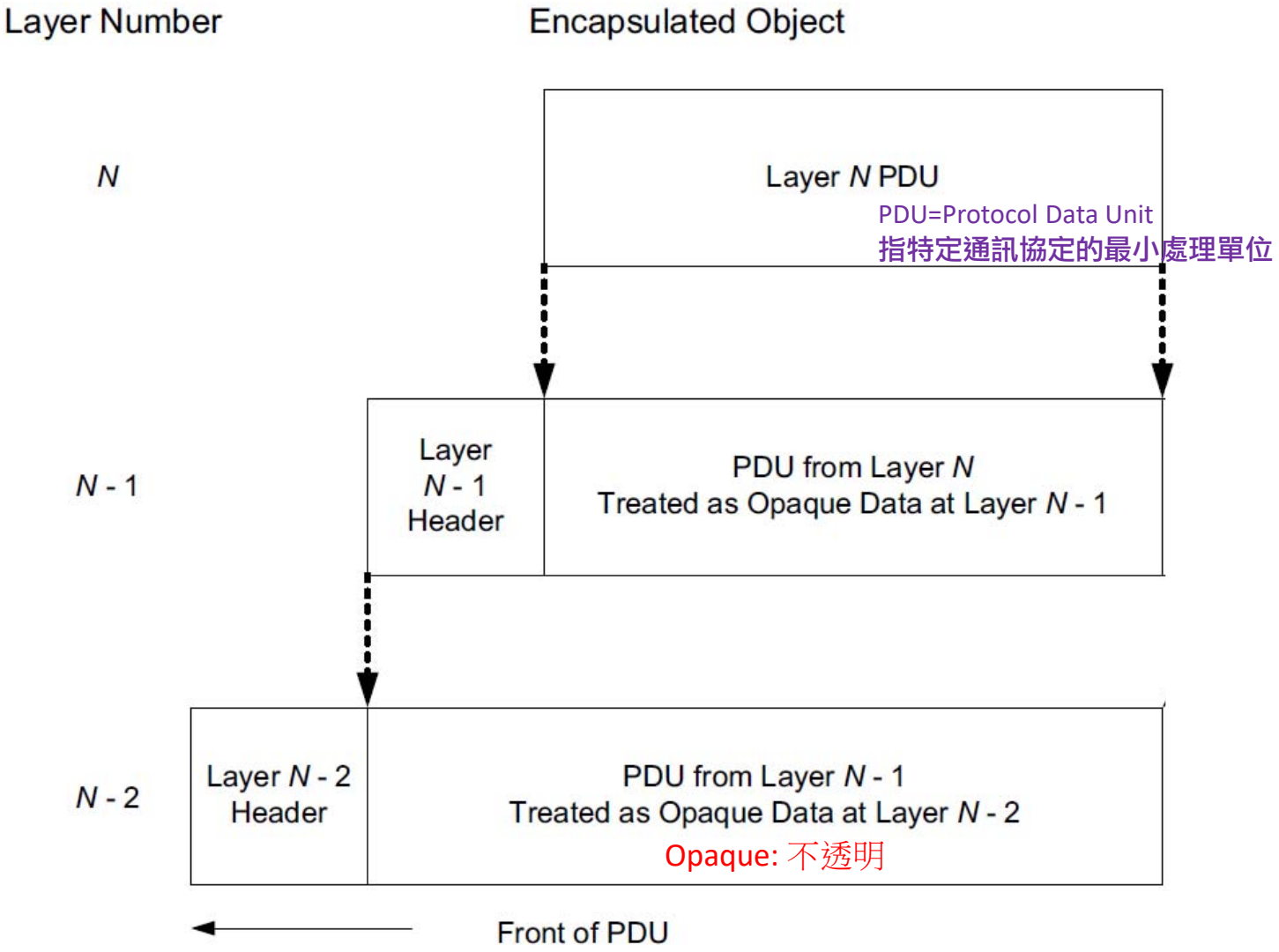
- Message

  軟體的訊息收發端稱為Endpoint

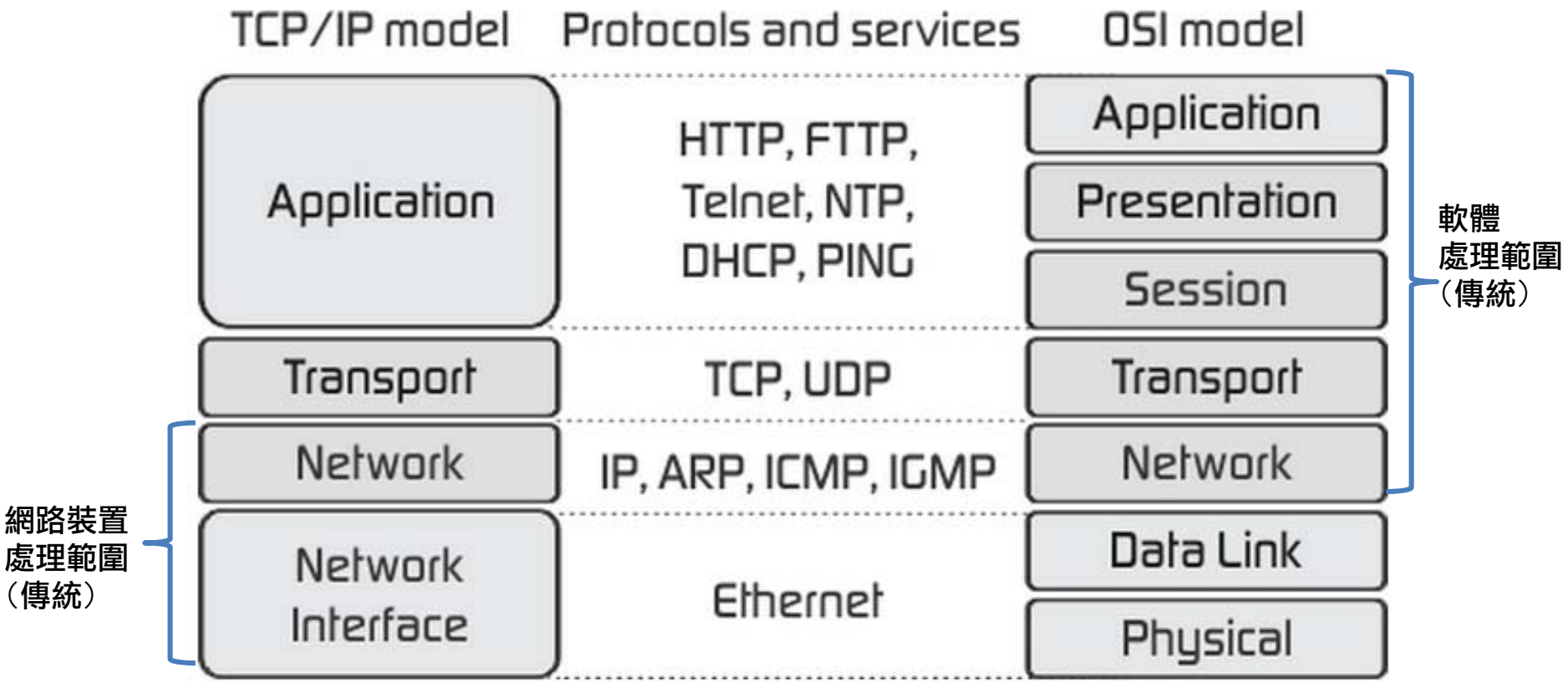
  – Logical data unit of communicating (endpoints' view point)
- Packet
  – The physical data unit of network transmission



封包分層處理概念


Layer Number
Encapsulated Object
N
Layer N PDU
PDU=Protocol Data Unit
指特定通訊協定的最小處理單位
N - 1
Layer N - 1 Header
PDU from Layer N
Treated as Opaque Data at Layer N - 1
N - 2
Layer N - 2 Header
PDU from Layer N - 1
Treated as Opaque Data at Layer N - 2
Opaque: 不透明
Front of PDU

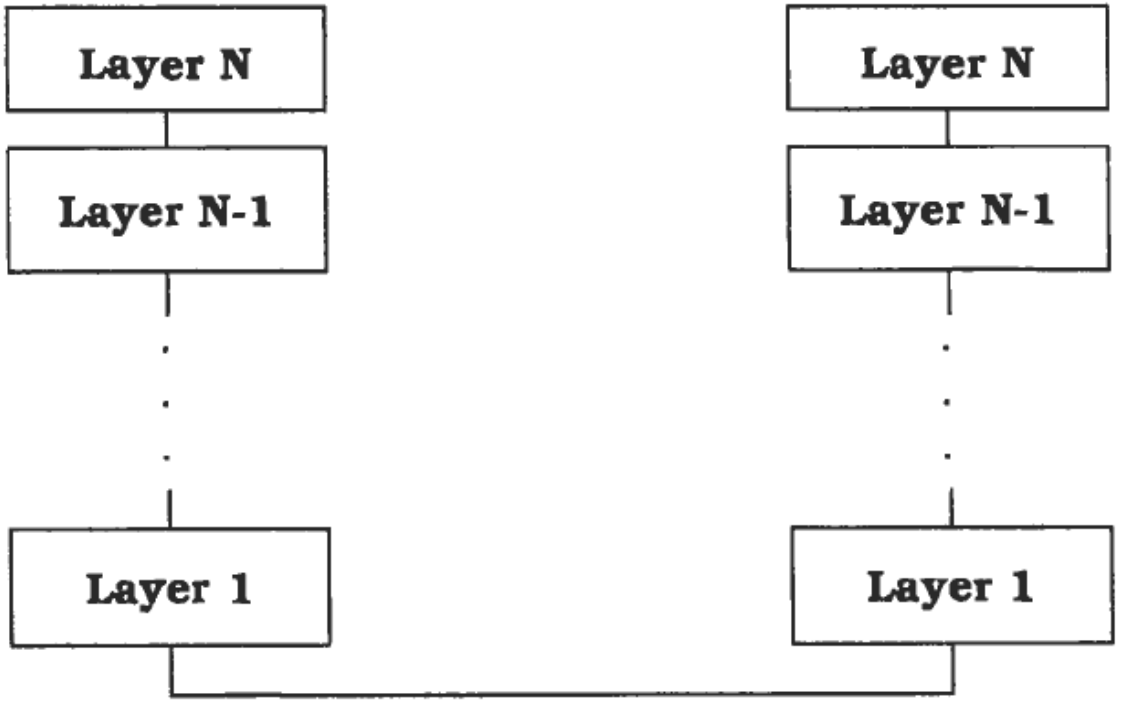
網路分層概念



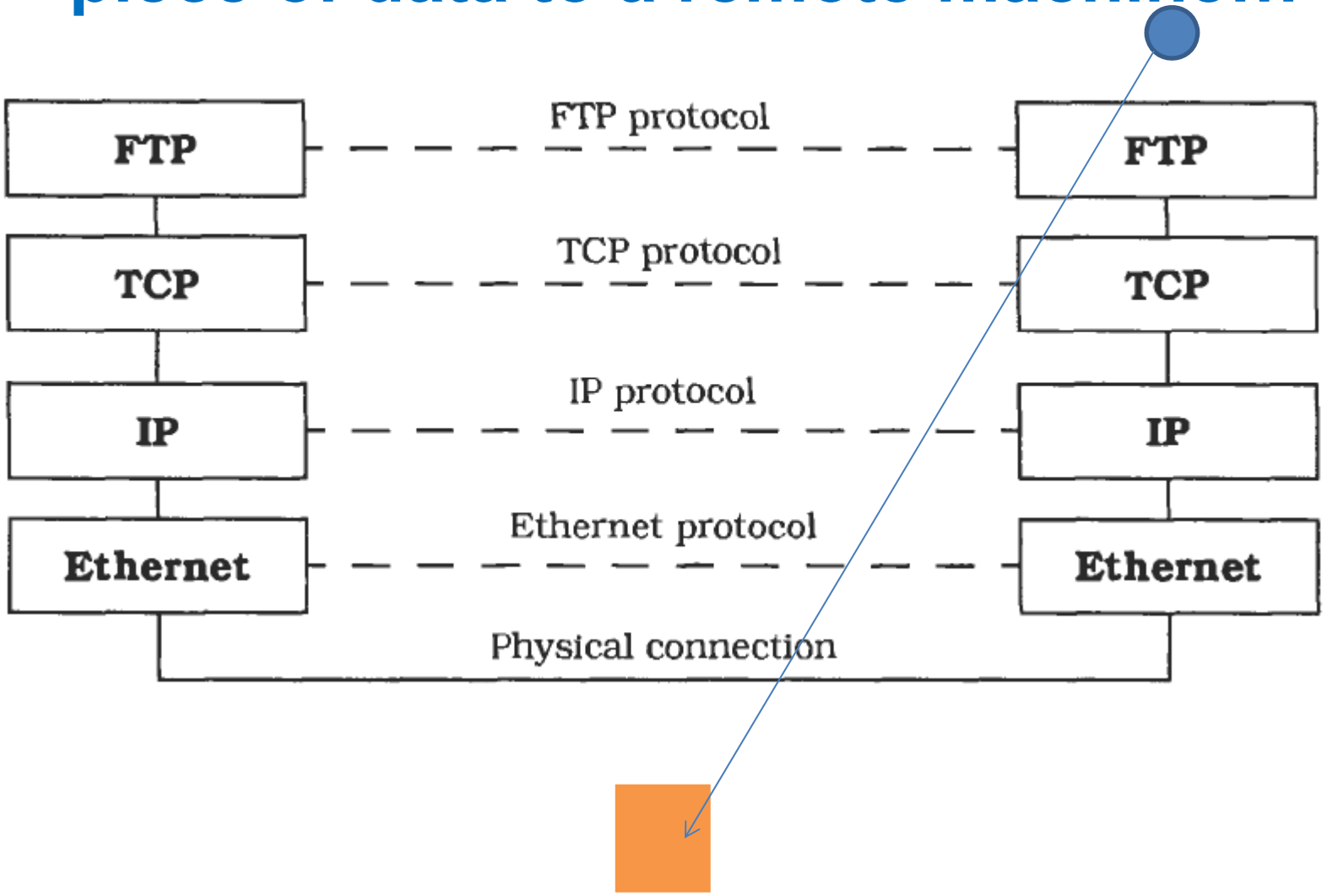
在SDN(Software Defined Network)的架構下，理論上所有層次均可用軟體處理

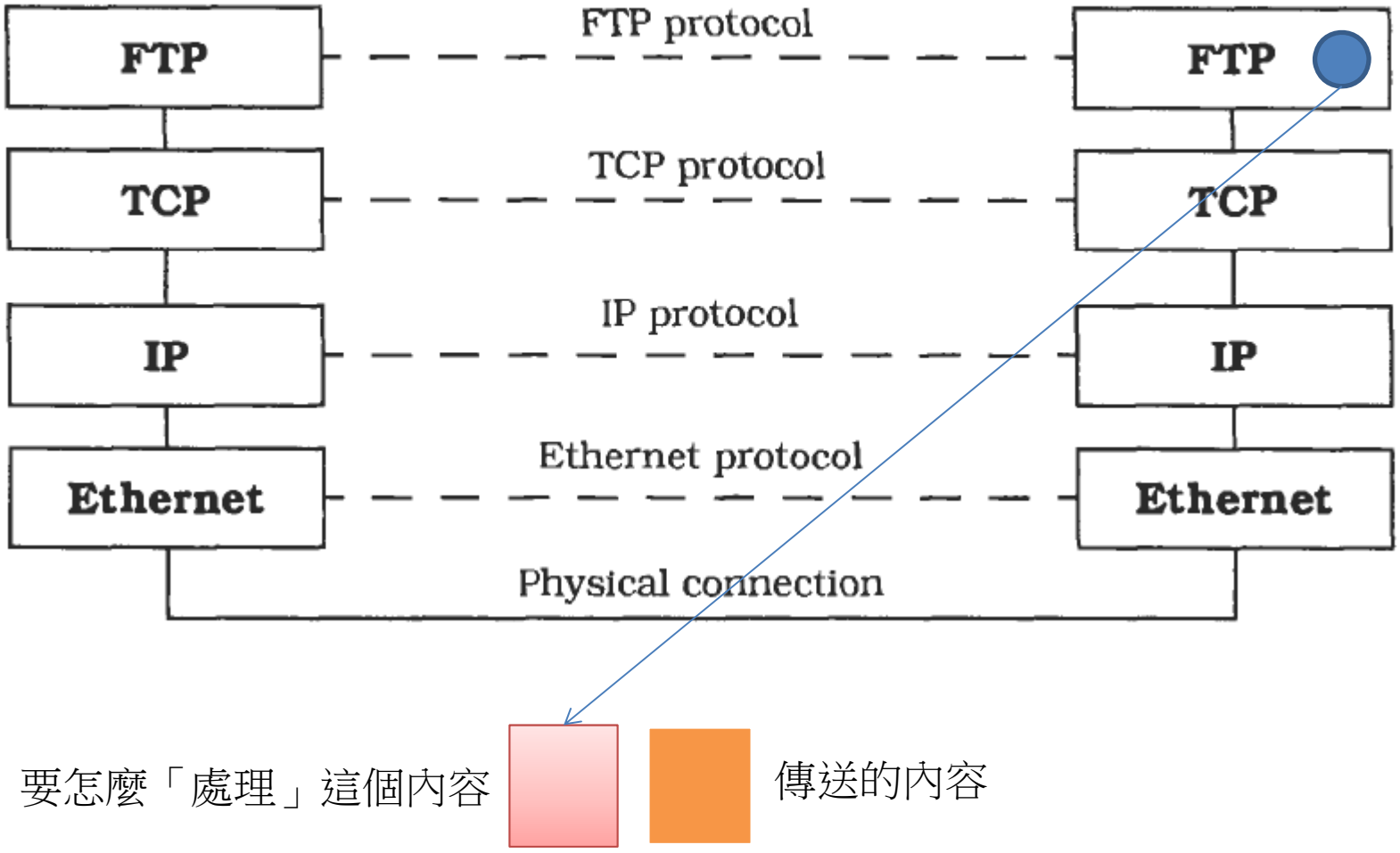
Layer

- Two-way Communication
  - Example: TCP/IP

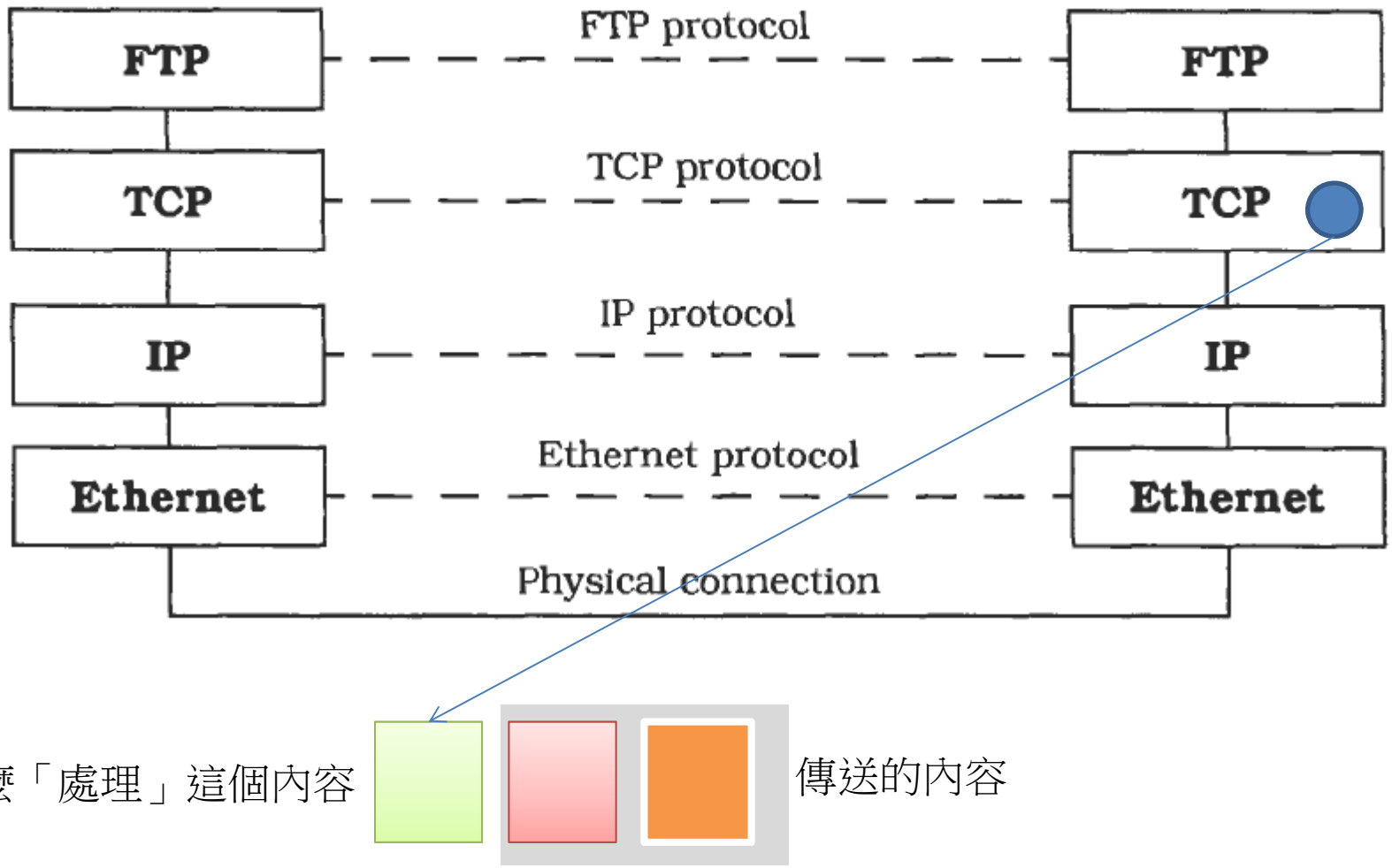


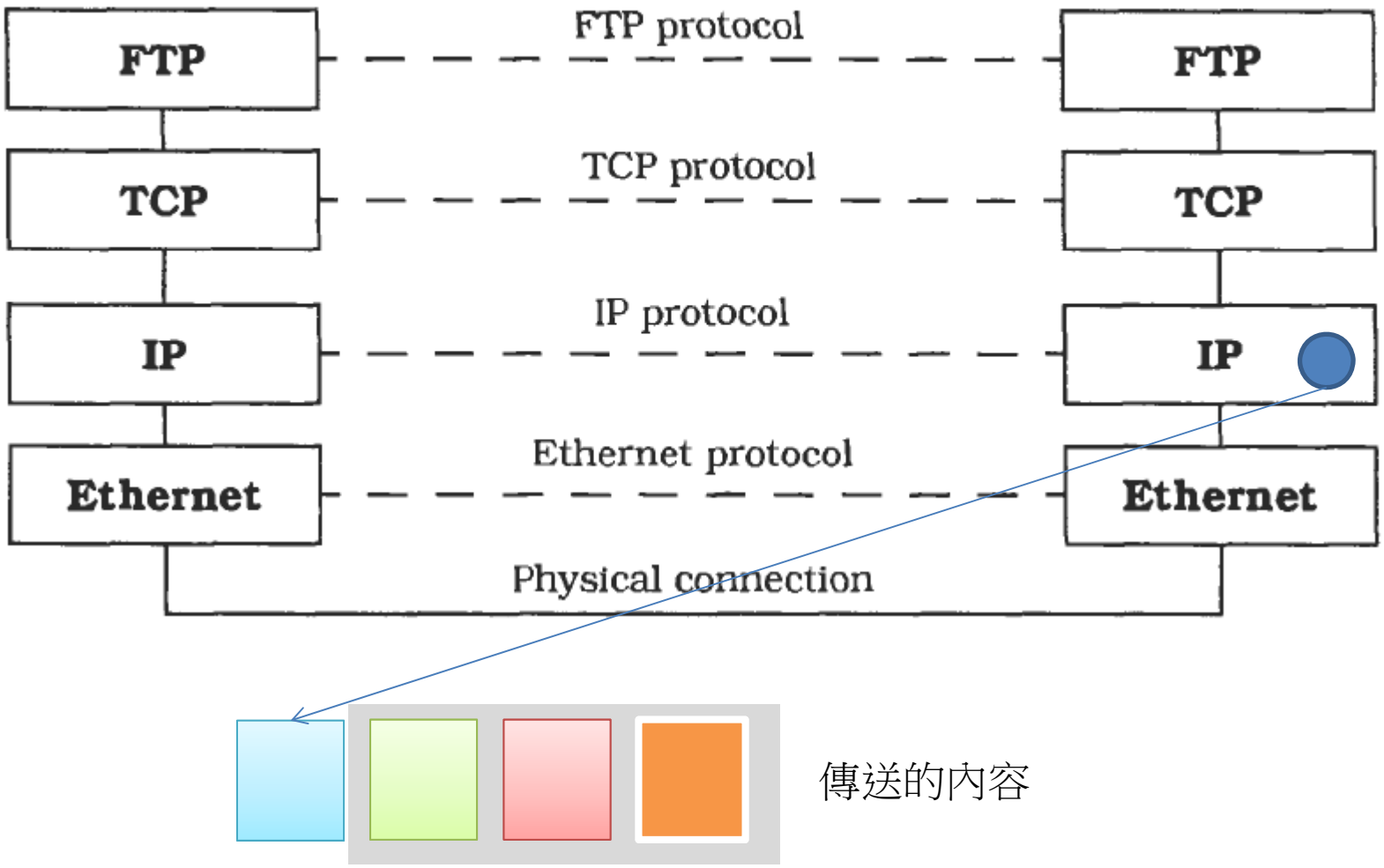
When an application wants to send a piece of data to a remote machine…

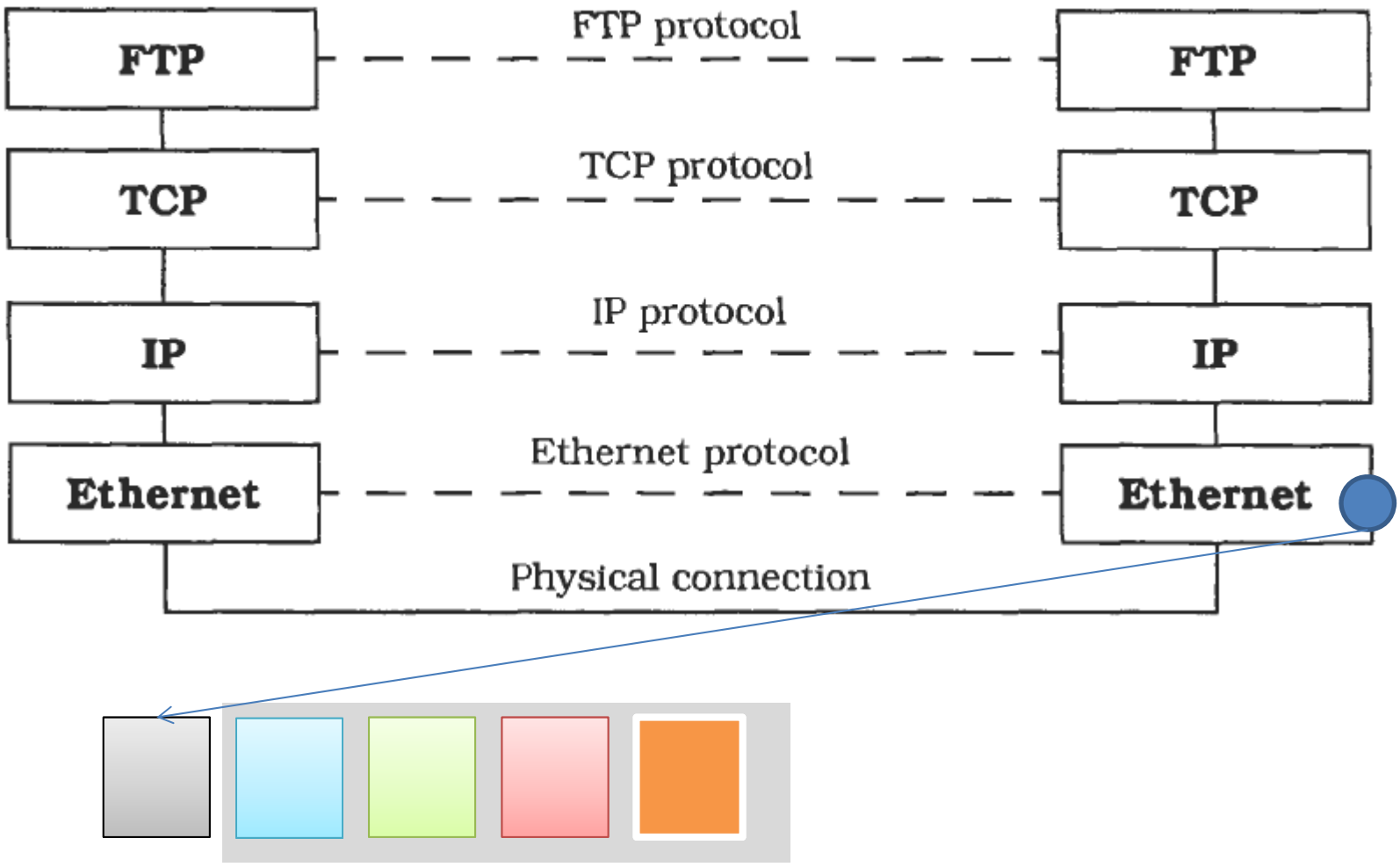




Add header of FTP

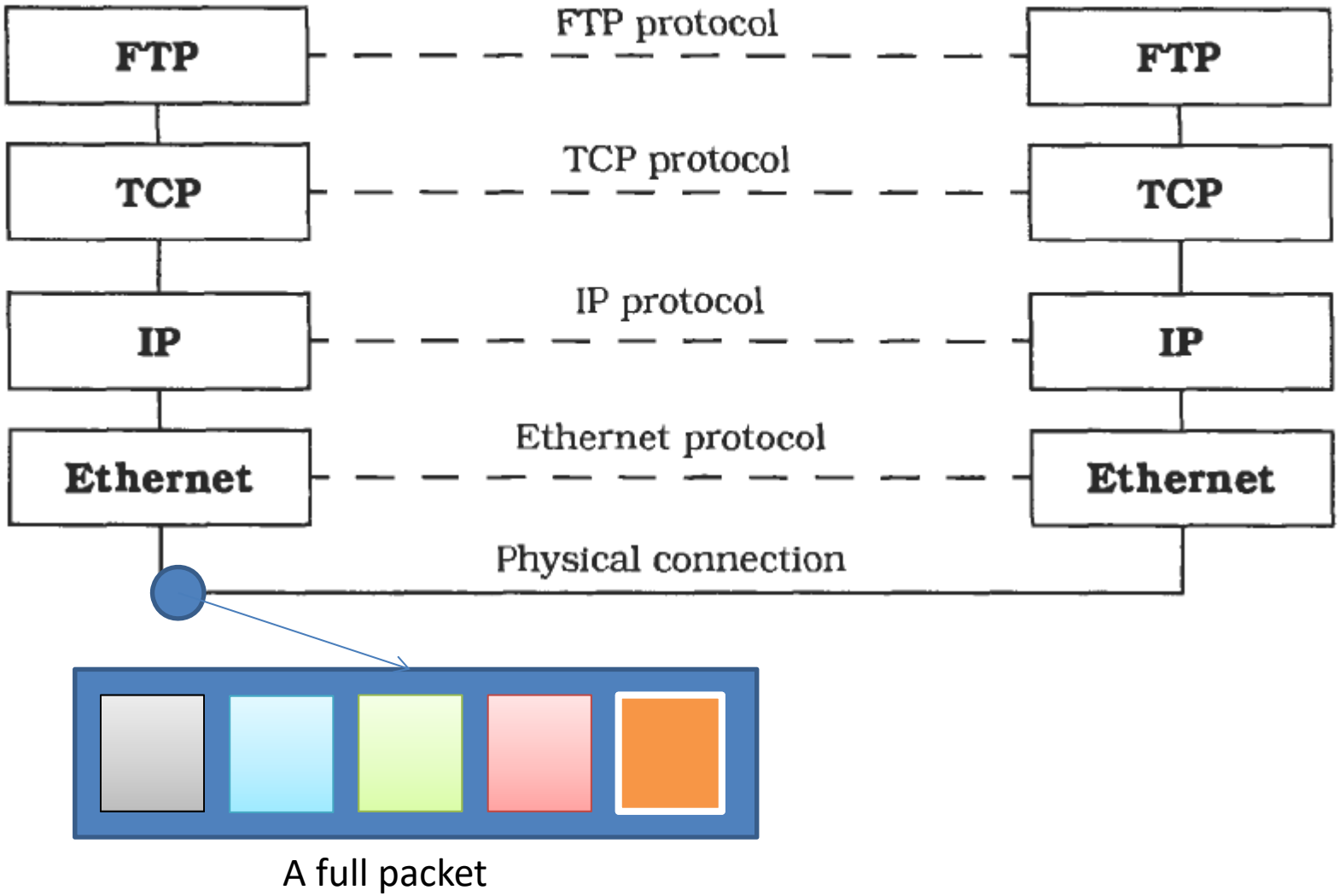

FTP
FTP protocol
FTP
TCP
TCP protocol
TCP
IP
IP protocol
IP
Ethernet
Ethernet protocol
Ethernet
Physical connection
要怎麼「處理」這個內容
傳送的內容
Add header of TCP

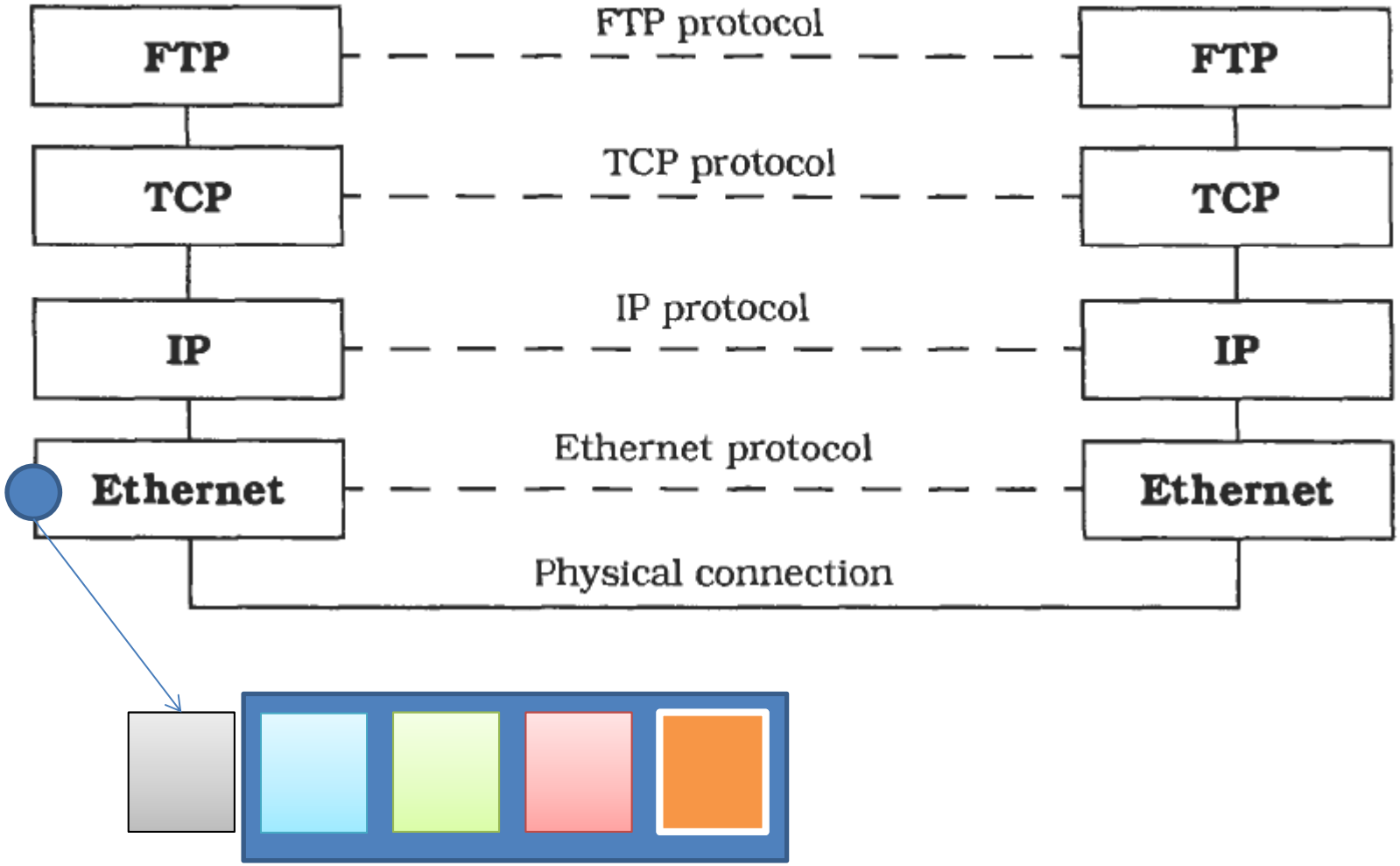

FTP
FTP protocol
FTP
TCP
TCP protocol
TCP
IP
IP protocol
IP
Ethernet
Ethernet protocol
Ethernet
Physical connection
傳送的內容
Add header of IP



Add header Ethernet

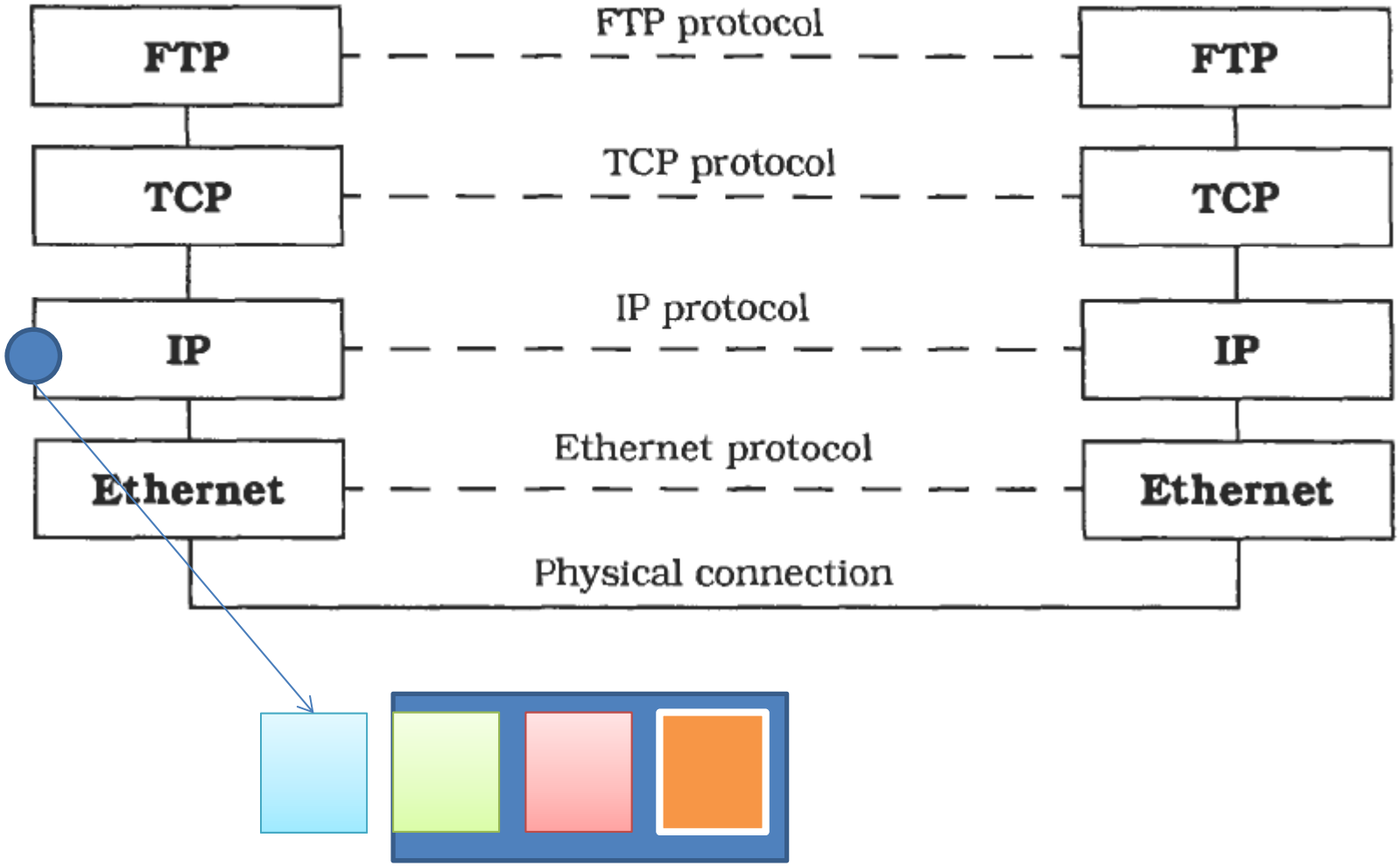
When a machine receives a packet…





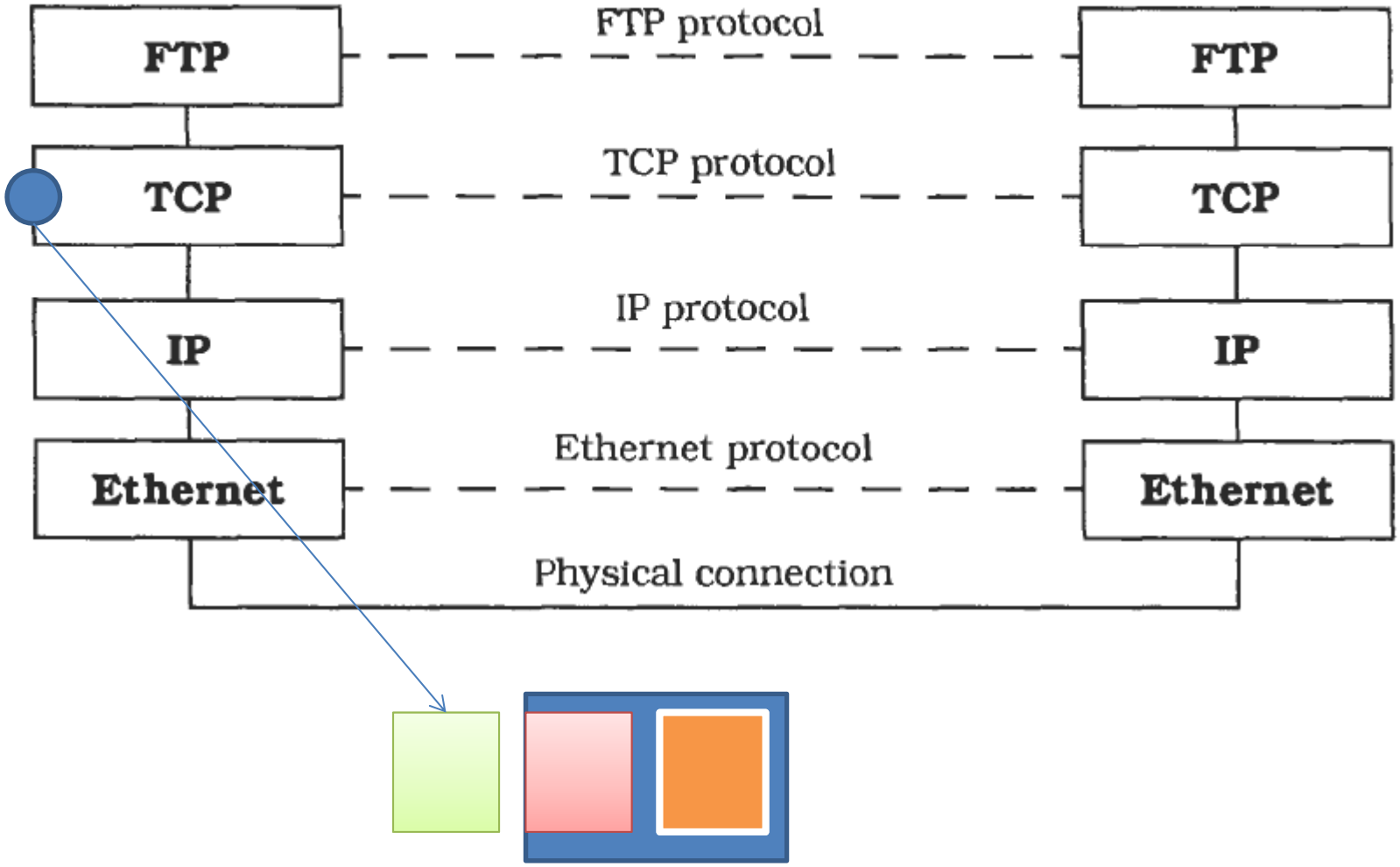
Processing header of Ethernet

依照header中的指示處理「內容」



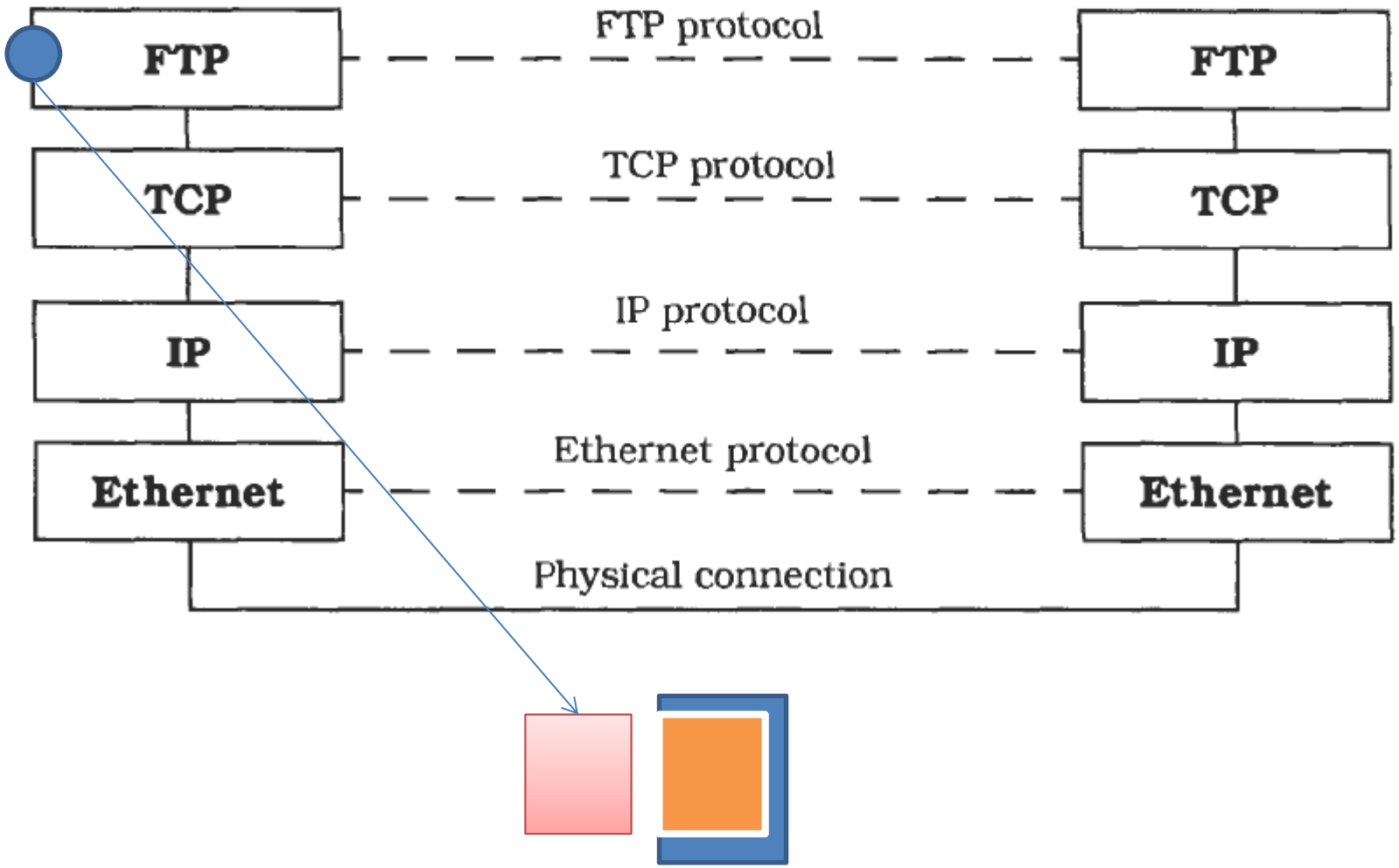
Processing header of IP

依照header中的指示處理「內容」



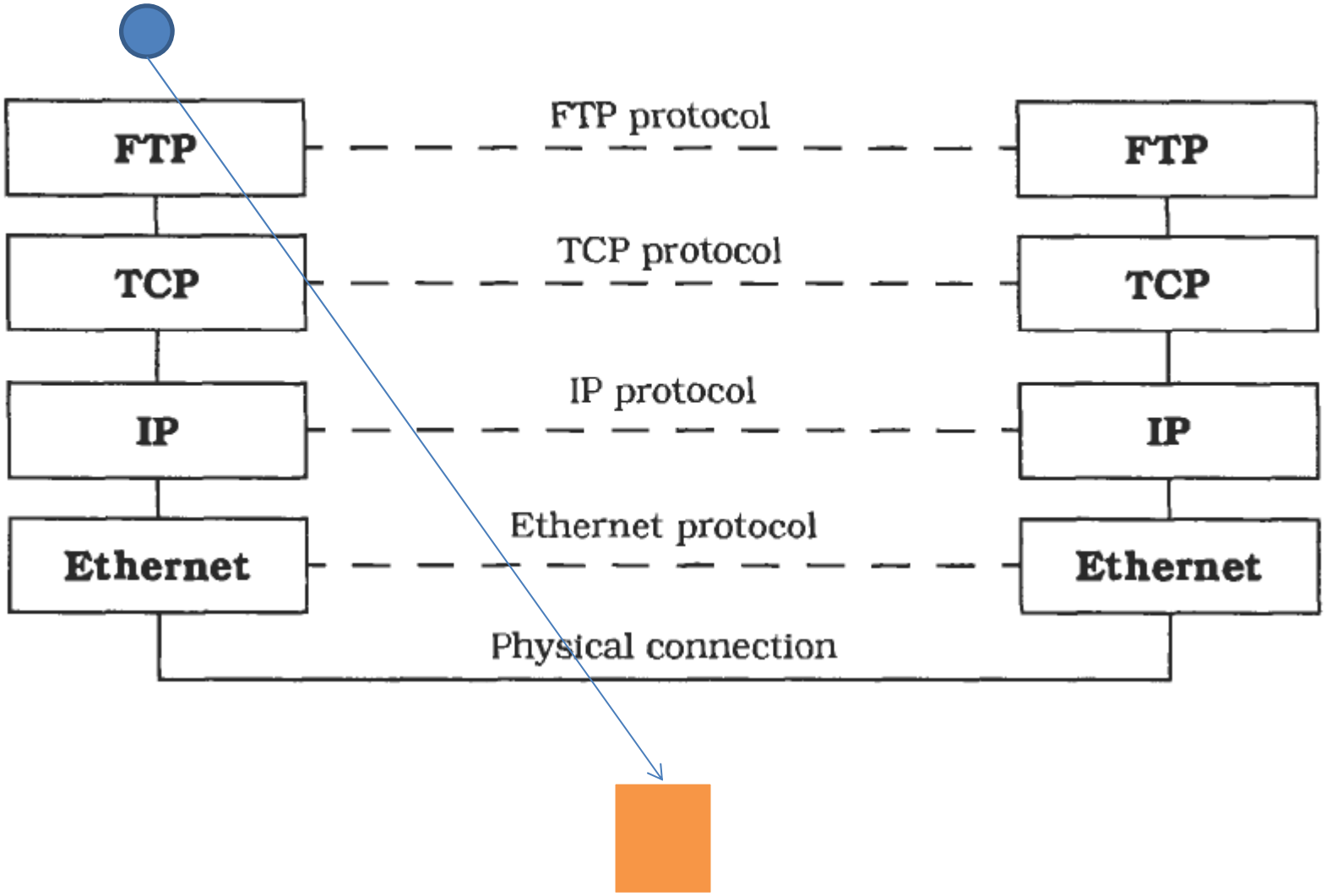
Processing header of TCP

依照header中的指示處理「內容」



Processing header of FTP

依照header中的指示處理「內容」

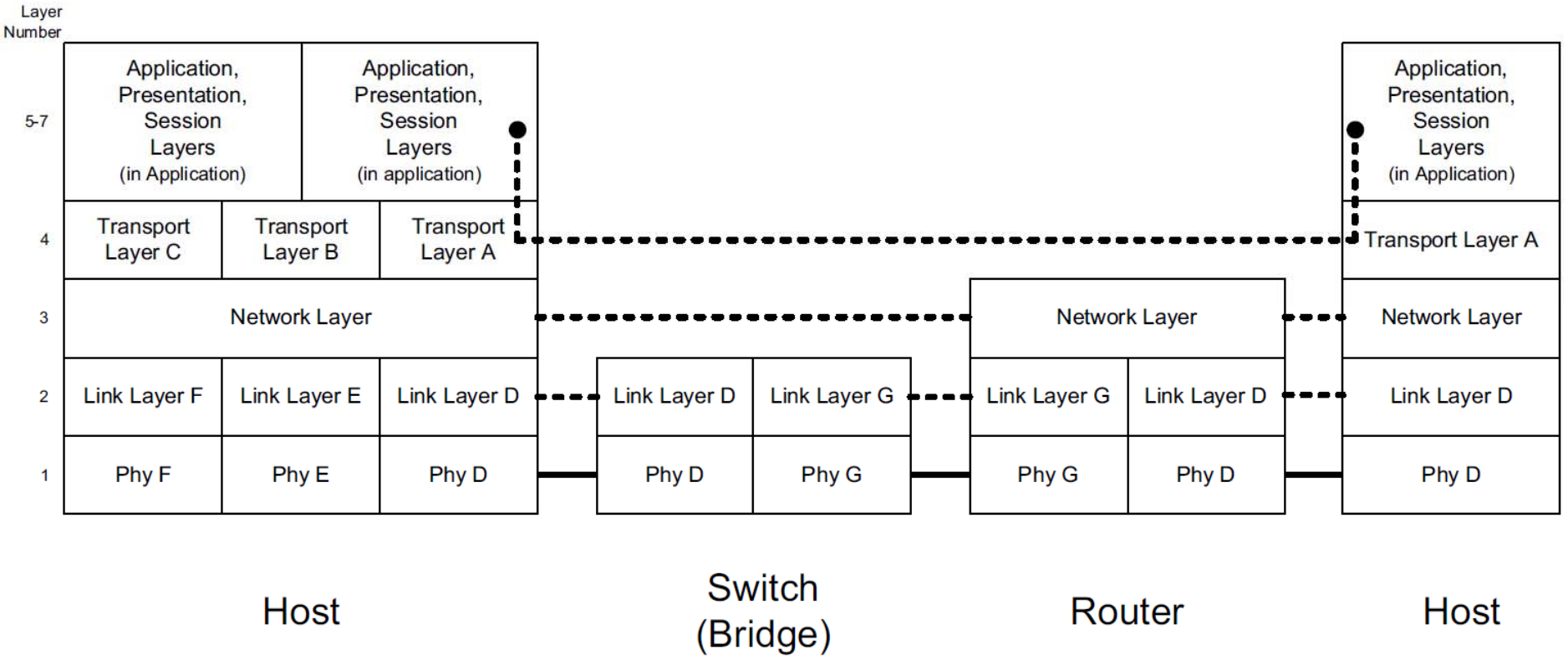


Finally, the application receives the data

網路分層概念

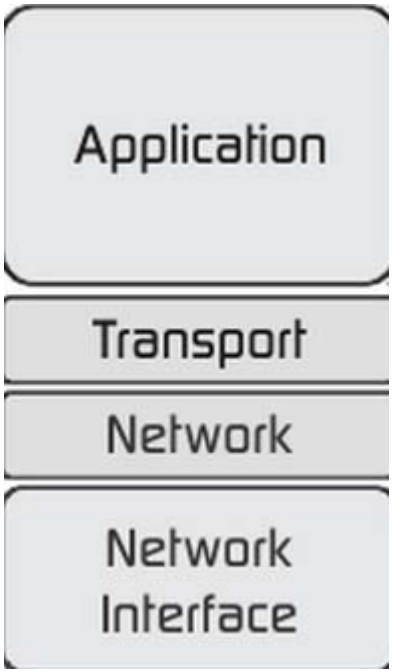
不一定每個端點都要解到AP層，有的網路設備只會解到「剛好需要」的層次，如switch等網路設備

- 例如一般 switch產品為layer 2 switch
- 處理到愈多的layers功能愈強，但價格愈貴，對效能影響也愈大



PDU

- 各式PDU (不同層次的資料單位)
  - Physical = Frame
  - Ethernet = Segment
  - TCP/UDP over IP = Datagram/Packet
  - HTTP = Message

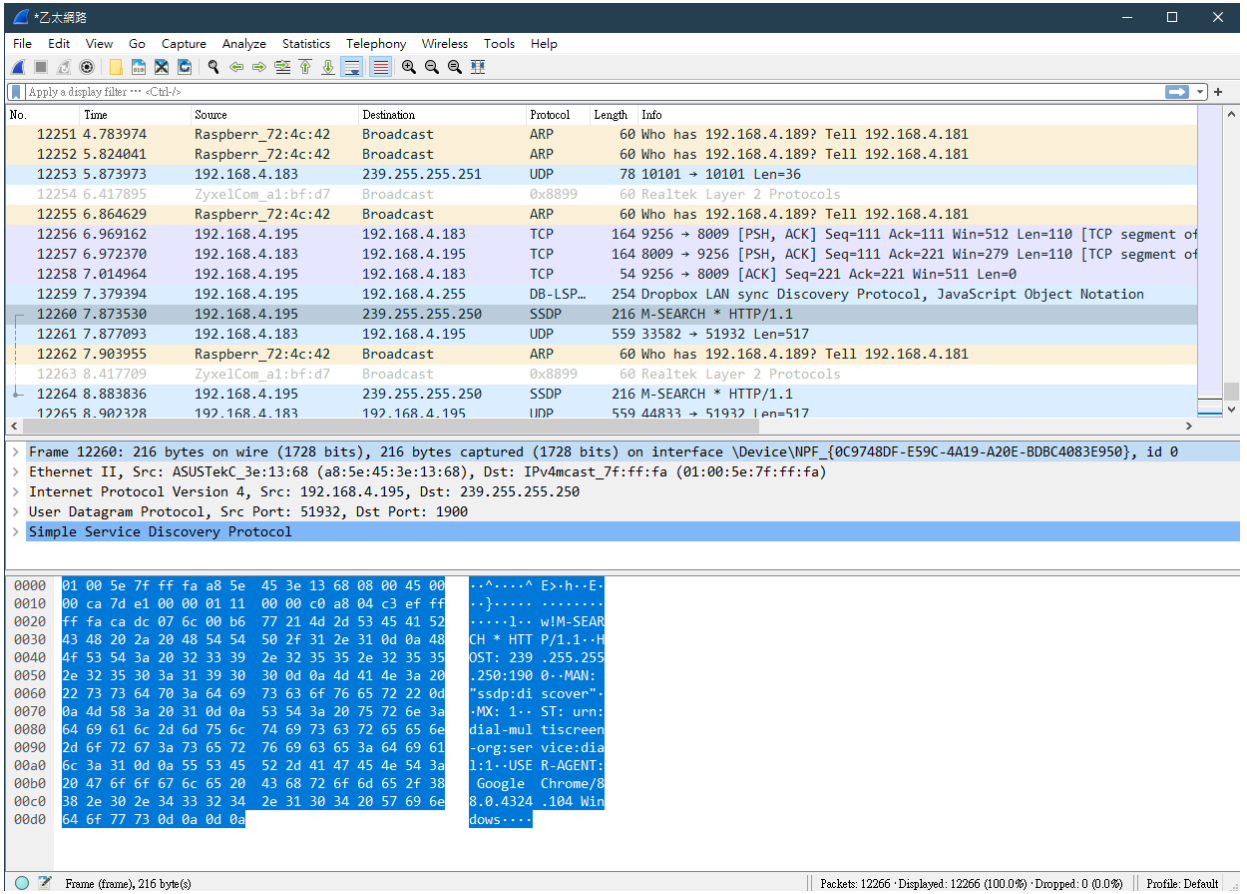


PDU
Protocol Data Unit

MTU
Maximum Transfer Unit
Ex:
MTU for Ethernet = 1500 bytes
MTU for IP = 64K bytes

Demo

- Wireshark
  - https://www.wireshark.org/
  - 練習觀察
    - Frame
    - Segment
    - Datagram
    - Message
    - 各層次的headers



IP 位址

- IPv4
  - 由四個連續0-255的數字構成，中間以「.」區隔
    - Ex: 140.119.1.110
    - 總共32 bits=4 bytes

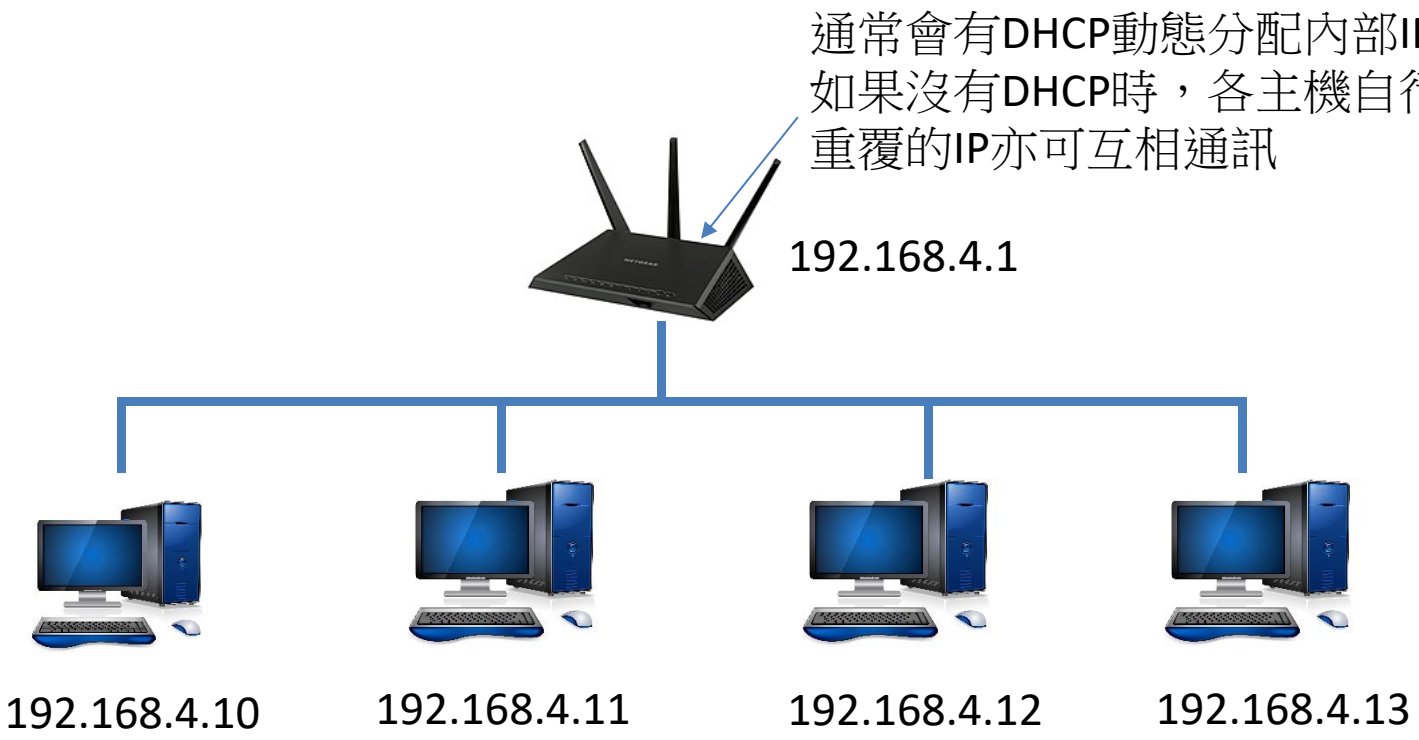
      10001100 01110111 00000001 01101110
  - 分為網段與主機段: 140.119.163.122 / 24 ←用來代表網段的長度
    網段愈大代表所轄IP數愈少
    - 140.119.163 代表 “政大資科系1”
      - 10001100 01110111 10100011 (共24 bits)
    - 122代表某台主機
      - 01111010

特殊網段

Prefix	Special Use	Reference
0.0.0.0/8	Hosts on the local network. May be used only as a source IP address.	[RFC1122]
10.0.0.0/8	Address for private networks (intranets). Such addresses never appear on the public Internet.	[RFC1918]
127.0.0.0/8	Internet host loopback addresses (same computer). Typically only 127.0.0.1 is used.	[RFC1122]
169.254.0.0/16	"Link-local" addresses—used only on a single link and generally assigned automatically. See Chapter 6.	[RFC3927]
172.16.0.0/12	Address for private networks (intranets). Such addresses never appear on the public Internet.	[RFC1918]
192.0.0.0/24	IETF protocol assignments (IANA reserved).	[RFC5736]
192.0.2.0/24	TEST-NET-1 addresses approved for use in documentation. Such addresses never appear on the public Internet.	[RFC5737]
192.88.99.0/24	Used for 6to4 relays (anycast addresses).	[RFC3068]
192.168.0.0/16	Address for private networks (intranets). Such addresses never appear on the public Internet.	[RFC1918]
198.18.0.0/15	Used for benchmarks and performance testing.	[RFC2544]
198.51.100.0/24	TEST-NET-2. Approved for use in documentation.	[RFC5737]
203.0.113.0/24	TEST-NET-3. Approved for use in documentation.	[RFC5737]
224.0.0.0/4	IPv4 multicast addresses (formerly class D); used only as destination addresses. 224.0.0.0–239.255.255.255	[RFC5771]
240.0.0.0/4	Reserved space (formerly class E), except 255.255.255.255.	[RFC1112]
255.255.255.255/32	Local network (limited) broadcast address.	[RFC0919] [RFC0922]

網段愈大代表所轄IP數愈少; 例如/32就只有一個IP

LAN

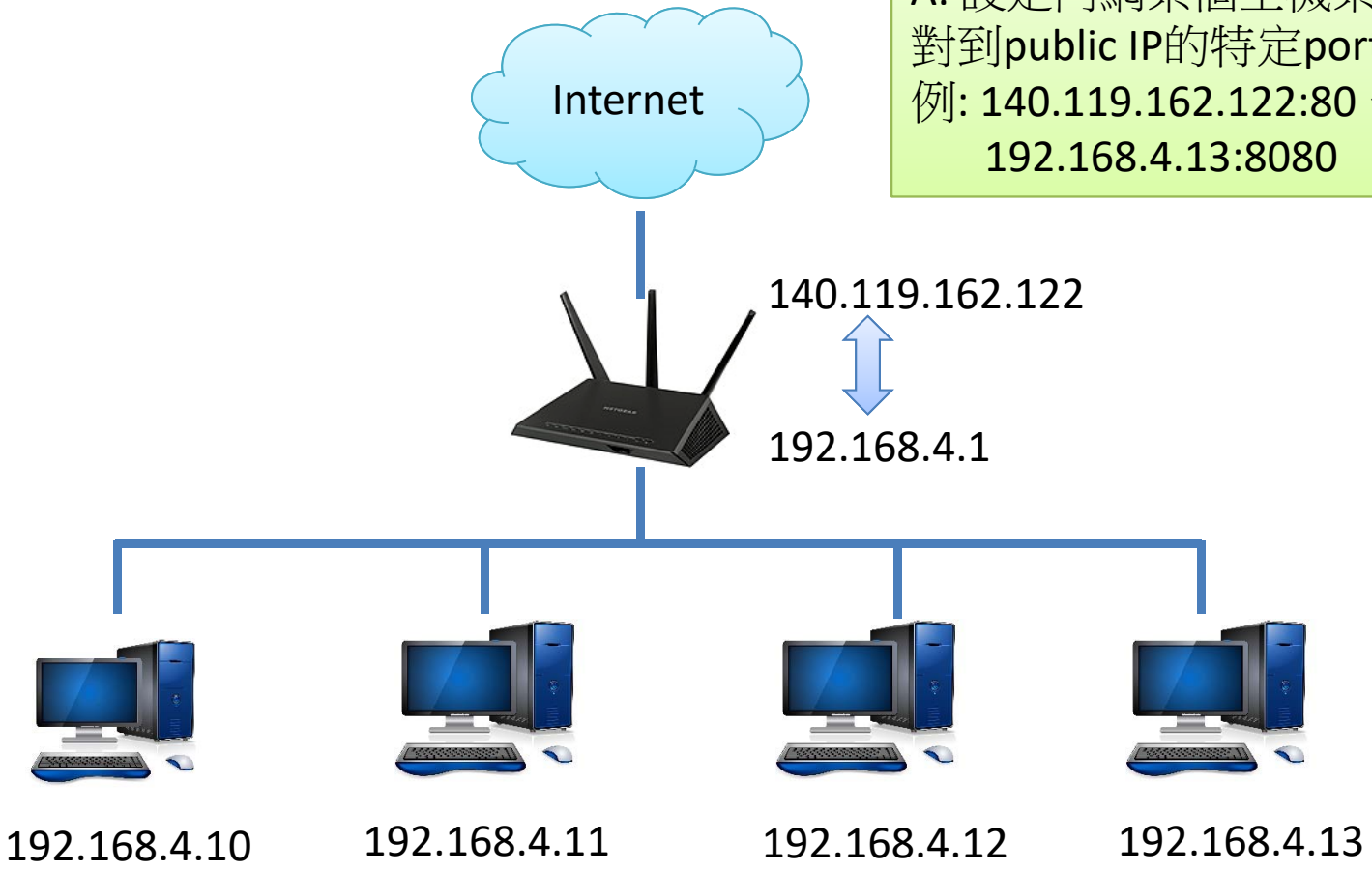


做實驗或Demo時常用配置，即使沒連到Internet，網內的各台主機還是可以互通

NAT

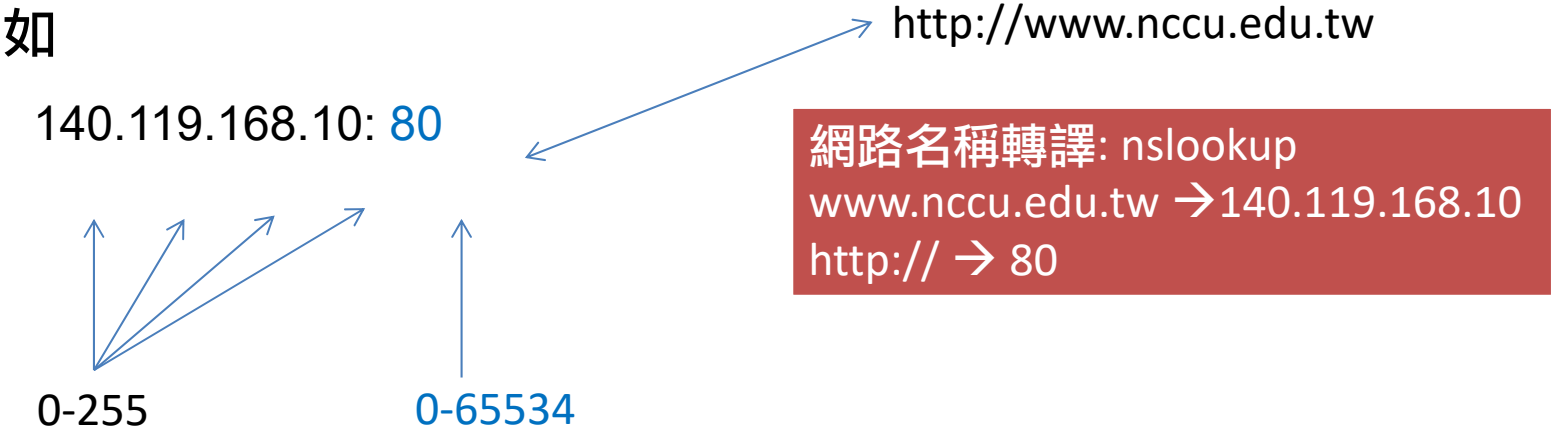
Q: 如何在網內架設public 服務?

A: 設定內網某個主機某個port
對到public IP的特定port
例: 140.119.162.122:80 ←→
192.168.4.13:8080



網路通訊

- 網路通訊程式
  - 位於二台不同電腦的程式互相傳送訊息
- Internet上每台電腦(的網卡)至少有一個固定位址
  - IP + Port
  - 例如
    - 140.119.168.10: 80

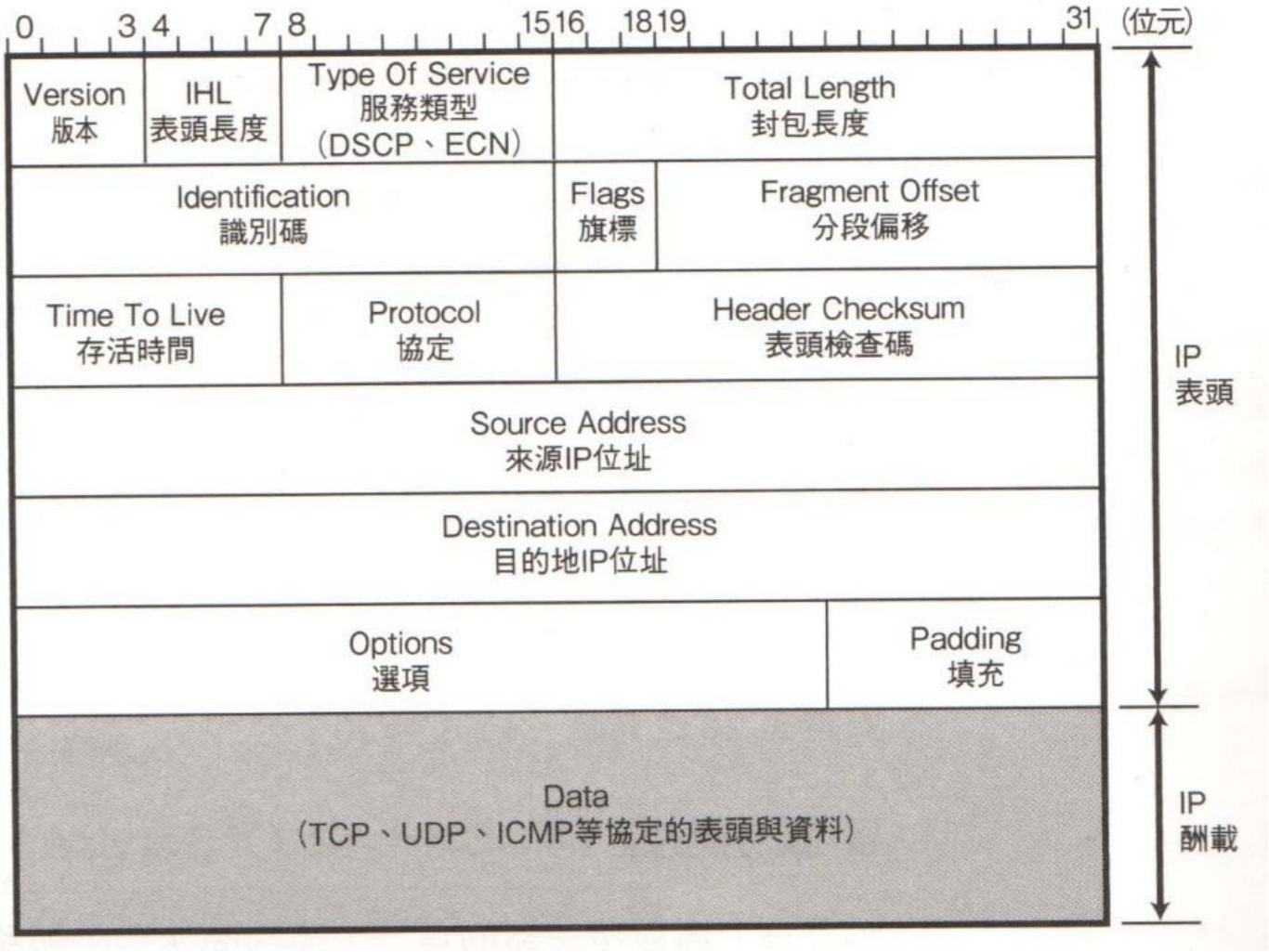


IP表頭只有IP的資訊，port是TCP/UDP層級的資訊

網路通訊

- 不同Port代表不同服務
- Well-known port
  - 80: HTTP
  - 443: HTTPS
  - 21: FTP
  - 22: SSH
  - 23: Telnet

IP 封包格式


0
3
4
7
8
15
16
18
19
31
(位元)
Version
版本
IHL
表頭長度
Type Of Service
服務類型
(DSCP、ECN)
Total Length
封包長度
Identification
識別碼
Flags
旗標
Fragment Offset
分段偏移
Time To Live
存活時間
Protocol
協定
Header Checksum
表頭檢查碼
Source Address
來源IP位址
Destination Address
目的地IP位址
Options
選項
Padding
填充
Data
(TCP、UDP、ICMP等協定的表頭與資料)
IP
表頭
IP
酬載

IP 封包格式

此分段位於原資料的第幾個位置
(單位=8bytes)

TOS目前實務上很少使用
目前亦有人用來做DSCP (Differentiated Service Codepoint, DiffServ)與
ECN(Explicit Congestion Notification)

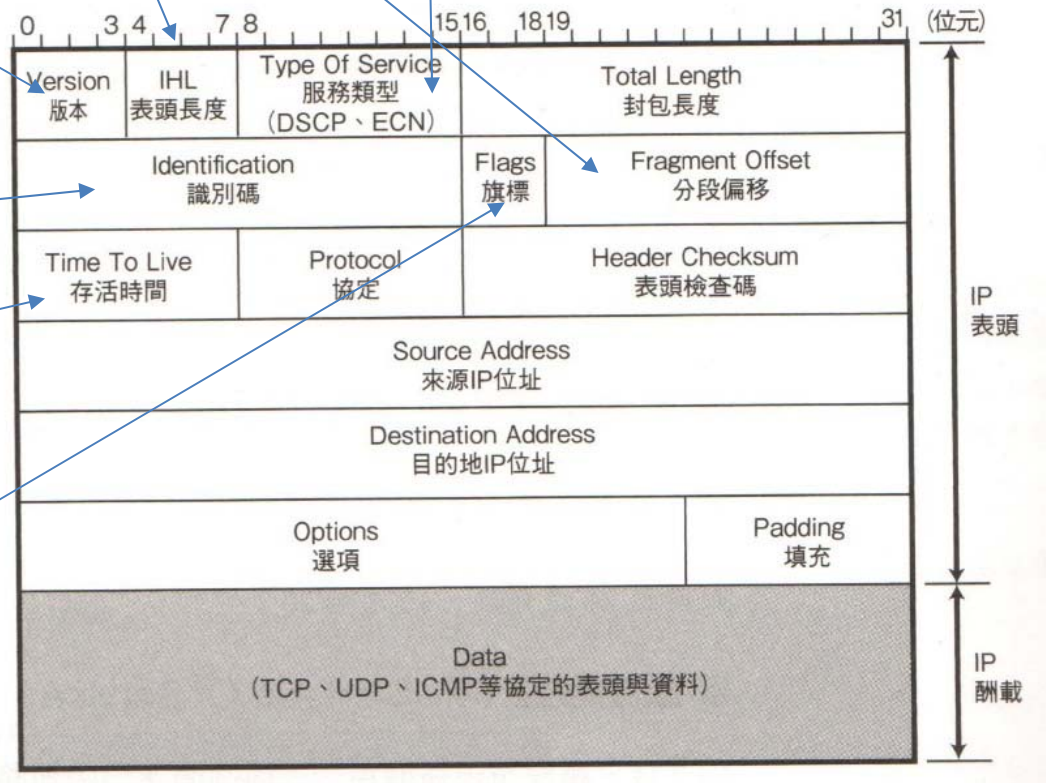
單位: 行(4 bytes)

4: IP
6: IPv6

分段，重組fragment用

可跨過多少中繼路由器

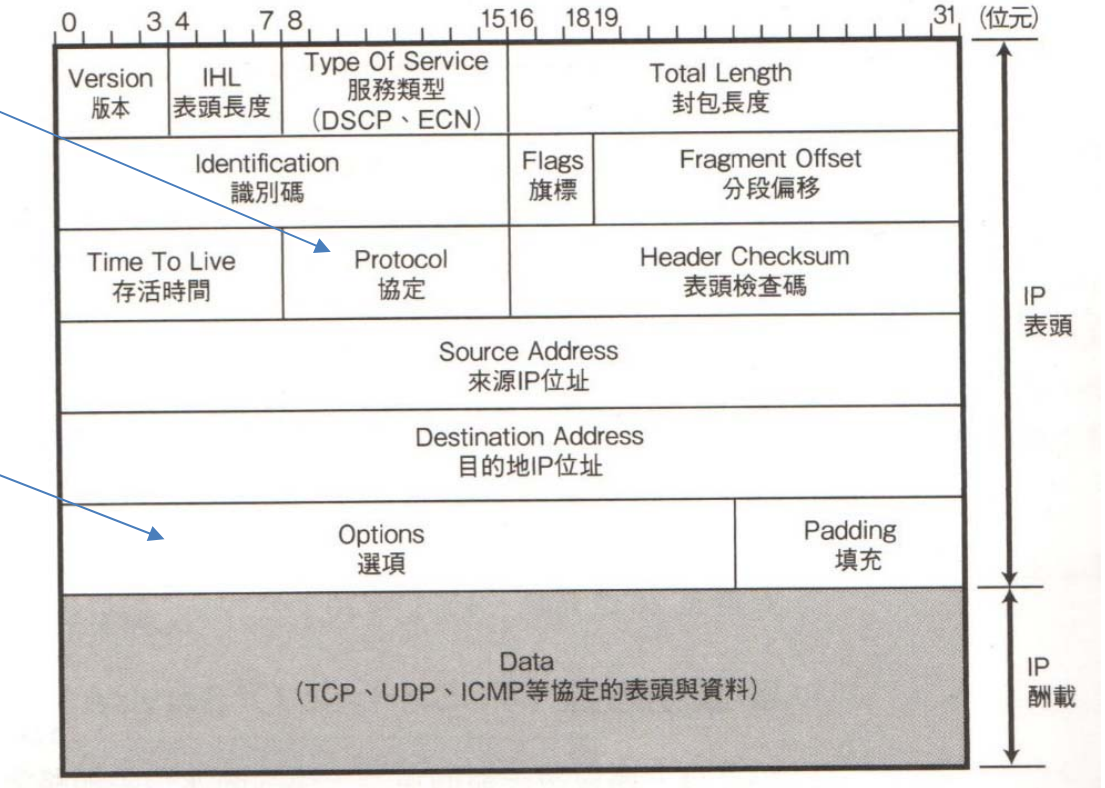
0: 一定要設成0
1: 是否分割(0可1不可)
2: 是否為最後的分割



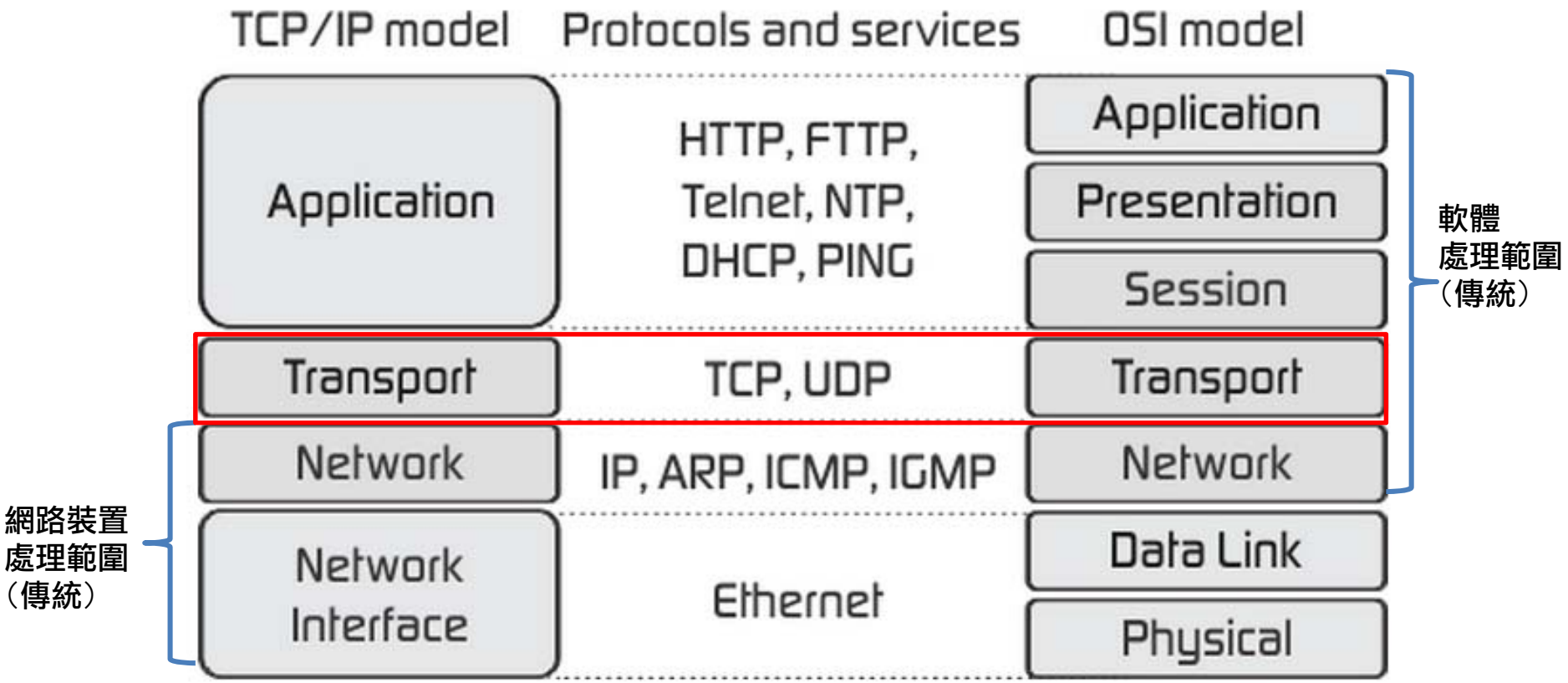
IP 封包格式

在payload中的協定代碼
TCP(6)
UDP(17)

一般不會使用，主要用於偵錯、測試



網路分層


TCP/IP model
Protocols and services
OSI model
Application
HTTP, FTTP,
Telnet, NTP,
DHCP, PING
Application
Presentation
Session
軟體
處理範圍
（傳統）
Transport
TCP, UDP
Transport
Network
IP, ARP, ICMP, IGMP
Network
網路裝置
處理範圍
（傳統）
Network
Interface
Ethernet
Data Link
Physical

User Datagram Protocol (UDP)

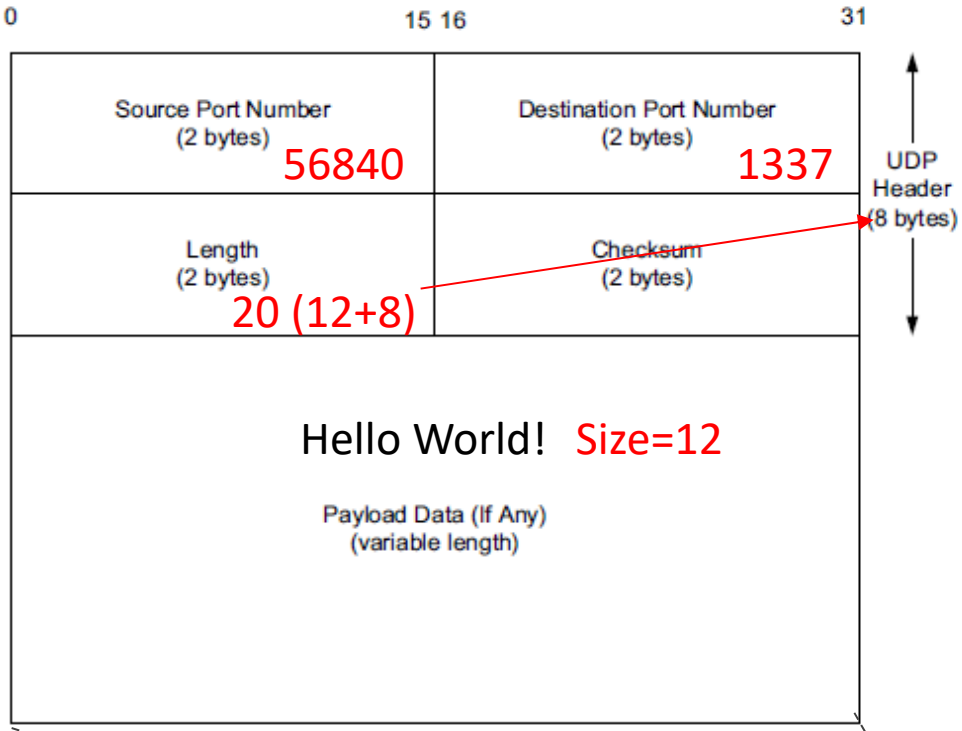
- 目的
  - 原意: 可按使用者(開發人員)需求客製化的協定
  - 快速、短、簡單的訊息傳送
  - 有基本的checksum機制(可得知封包是否損壞)
- 限制 (缺乏TCP的主要功能)
  - No session
  - No duplicate protection
  - No delivery guarantee
  - No order guarantee
  - No traffic control

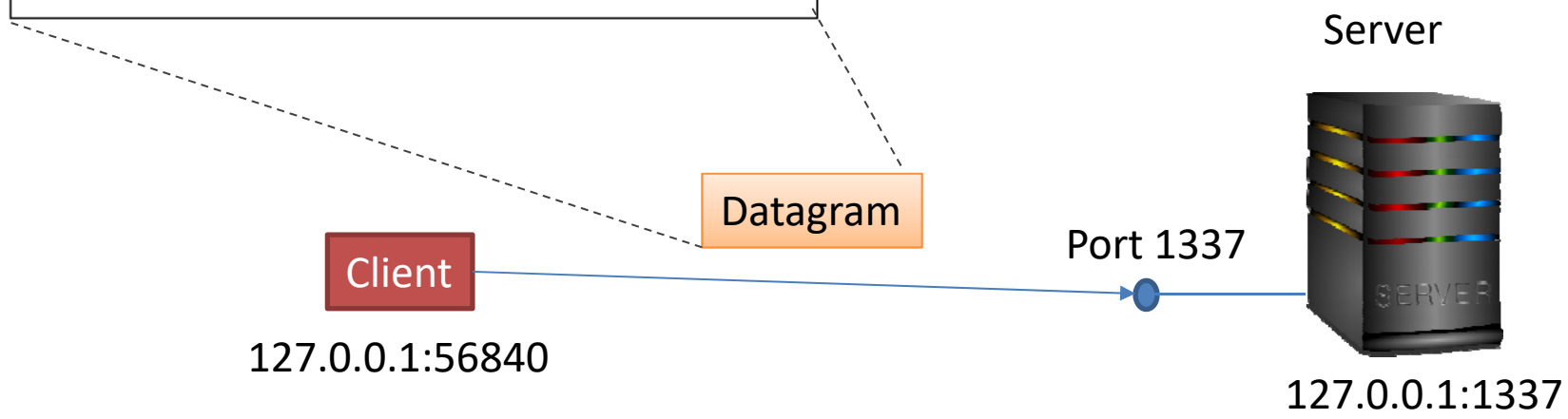
可能的適用場合
Heartbeat
Service discovery
LLN (Low power and lossy network 低功耗網路)
Idempotent operations: 媒體群播、DNS查詢

User Datagram Protocol (UDP)

- 客製化範例
  - QUIC (初名: Quick UDP Internet Connection)
    - 最早由Google Jim Roskind 所發展
    - 提供幾乎等同於HTTP over TCP連線的可靠度，但延遲大大減少
      - 在高層級(應用程式層)處理封包異常、遺失問題; 請求間不互相影響
    - 後來成為IETF標準，並成為HTTP/3的基礎
      - IETF版本和Google原始提交版已有重大差異
  - Google Nest Thread
    - IPv6-based, low-power mesh networking technology for IoT
    - Over UDP/6LowPAN/802.15.4
  - GPRS Tunneling Protocol (GTP)
    - IP-based protocol used to carry GPRS within GSM, UMTS, LTE and 5G

User Datagram Protocol (UDP)





Node.js 的UDP API

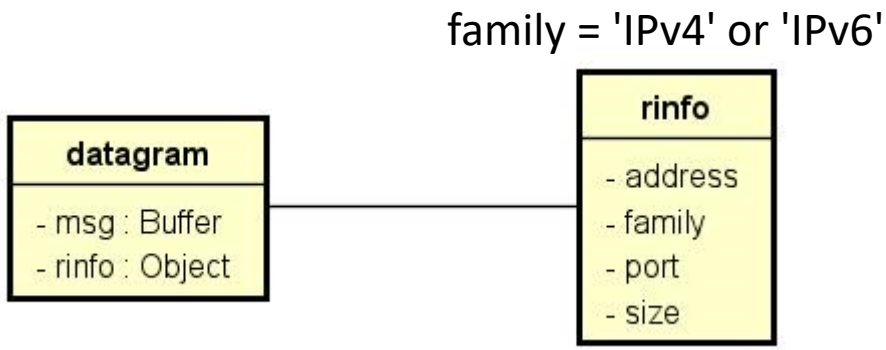
- Module name
  - UDP/datagram
- Usage
  - const dgram = require(‘dgram’);
  - const udpSocket = dgram.createSocket(‘udp4’);
  - **傳送訊息**

```
udpSocket.send(msg, port, ‘IP’, callback);
```

Text
function
Buffer
Number

  - **接收訊息**

```
udpSocket.on(‘message’, function (msg, rinfo) {
   …
});
```



https://nodejs.org/dist/latest-v18.x/docs/api/

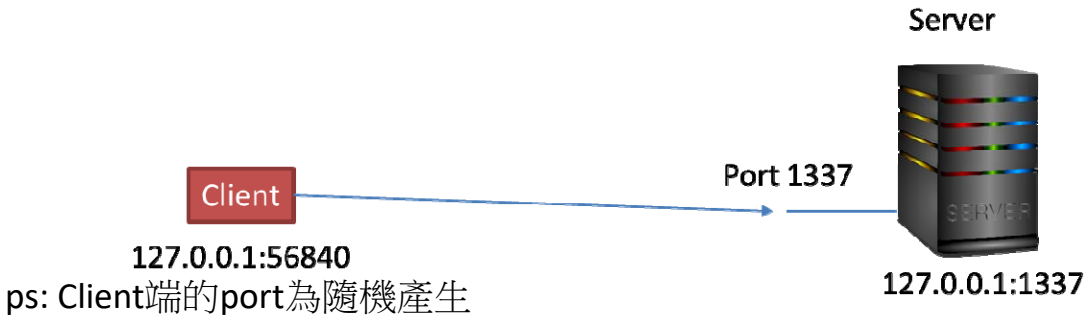
Buffer的處理

- 資料轉換為Buffer
  - 資料要在網上傳送前，要先轉為Buffer
  - Buffer.from(x)
    - x建議為string
      - x若為物件，先用JSON.stringify(x)轉為string
- Buffer轉為資料
  1. 使用buffer.toString()
  2. 還原為物件: JSON.parse()

```
let obj = {
   key0: 'value0',
   key1: 'value1',
   key2: 'value2',
   key3: 'value3',
   key4: 'value4'
};

// 傳送端: 先將json轉為string，再轉為buffer
let buf = Buffer.from(JSON.stringify(obj));
console.log(buf);

//  接收端: 先將buffer轉回string，再還原為json
let obj1 = JSON.parse(buf.toString());
console.log(obj1);
```

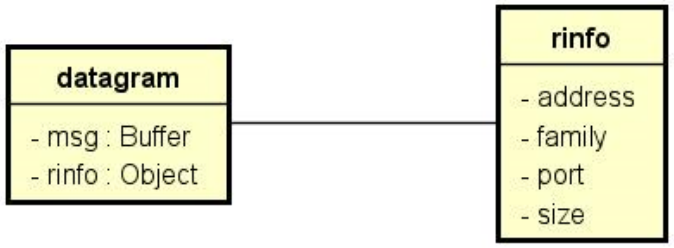
Node.js UDP Client

```
const dgram = require('dgram');
const client = dgram.createSocket('udp4');

const message = Buffer.from('Hello World!');

client.send(message, 1337, 'localhost', (err) =>
{
   client.close();
});
```

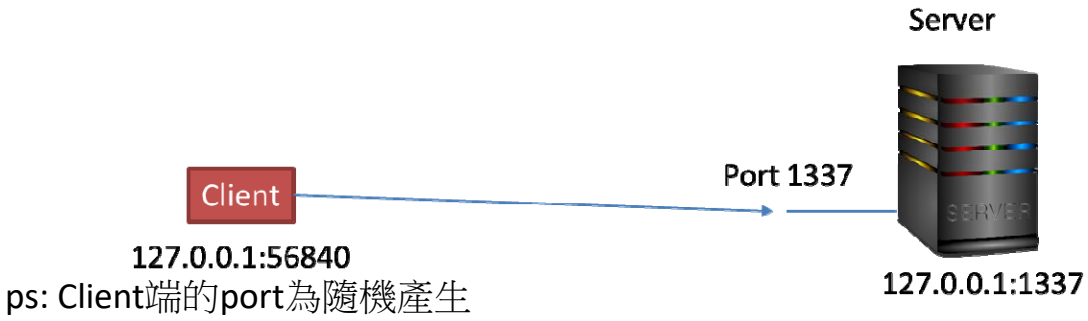


Node.js UDP Server

```
const dgram = require('dgram');
const server = dgram.createSocket('udp4');

server.on('message', (msg, rinfo) => {
    ....處理與回應訊息...
});

server.on('error', (err) => {
    ...處理錯誤...
});

server.bind(1337);
```



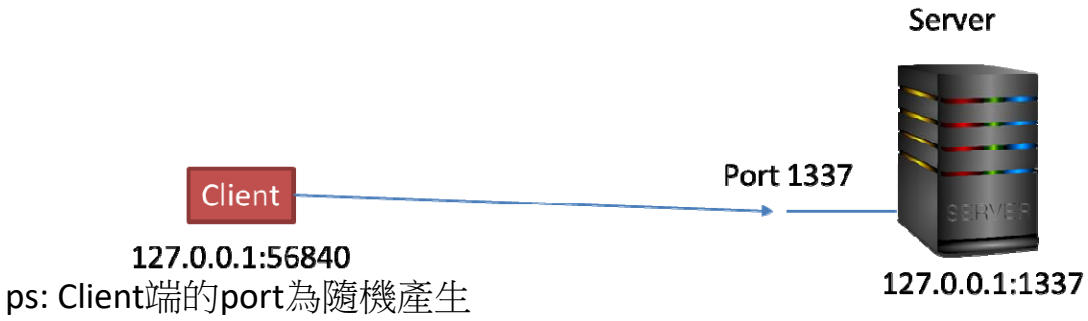
msg是Buffer型別，
所以若傳送進來的訊息是JSON物件，
可使用
let obj = JSON.parse(msg.toString());
來取得物件



Python UDP Client

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("127.0.0.1", 1337))
msg = b"Hello UDP Server"
s.send(msg)
s.close()
```

要先轉成byte才能送出



Python UDP Server

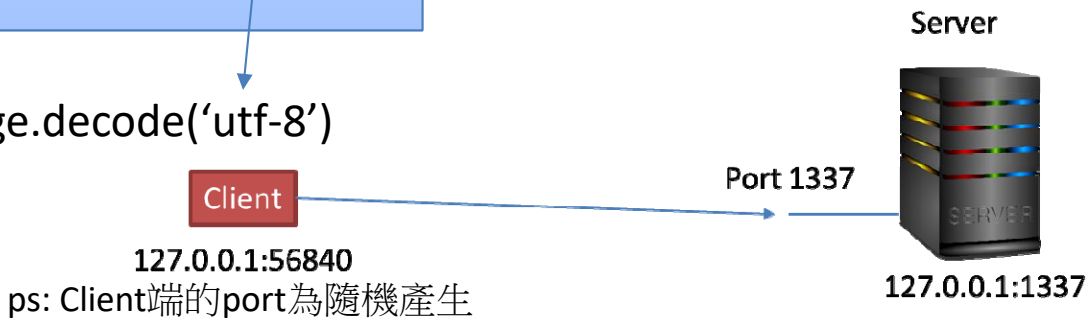
```
import socket

ss = socket.socket(family=socket.AF_INET,
                   type=socket.SOCK_DGRAM)
ss.bind(("127.0.0.1", 1337))

while True:
    message, address = ss.recvfrom(1024)
    clientMsg = f"Message from Client:{message}"
    clientIP = f"Client IP Address:{address}"
    print(clientMsg)
    print(clientIP)
```

指定使用IPv4

指定使用UDP

最大buffer size =1024

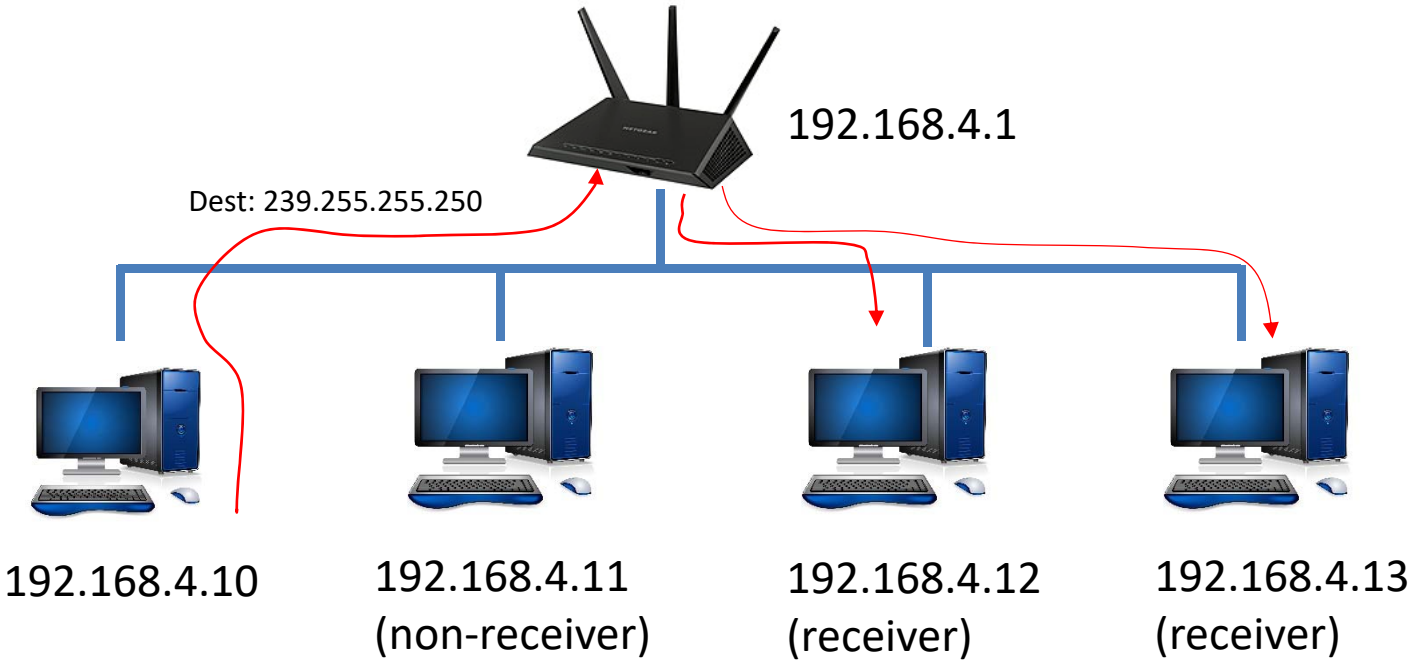
Binary to text的方法: message.decode('utf-8')



UDP Multicast

進行IP Multicast
Sender會送訊息到某個class D的multicast address (224.0.0.0–239.255.255.255間的其中一個)
Receiver向switch(router)訂閱同一個multicast address (add membership)

Router二種機制:
With IGMP:
　Router在接收到Sender送到該multicast address的封包後派送到有訂閱的receiver
Without IGMP:
　廣播封包到所有主機，由該主機網路卡過濾是否接收封包



Multicast Sender

同一台機器，如果需要多個receiver的話，要多個IP 要bind到239.255.255.250

```
let dgram = require('dgram');
let server = dgram.createSocket({type: 'udp4', reuseAddr: true});

server.bind(2391, () => {
    setInterval(() => {    每5秒送一次訊息
        server.send('Test', 2390, "239.255.255.250");
    }, 5000);
});

server.on('message', function (msg, rinfo) {
    …
});
```

回應時的處理方式

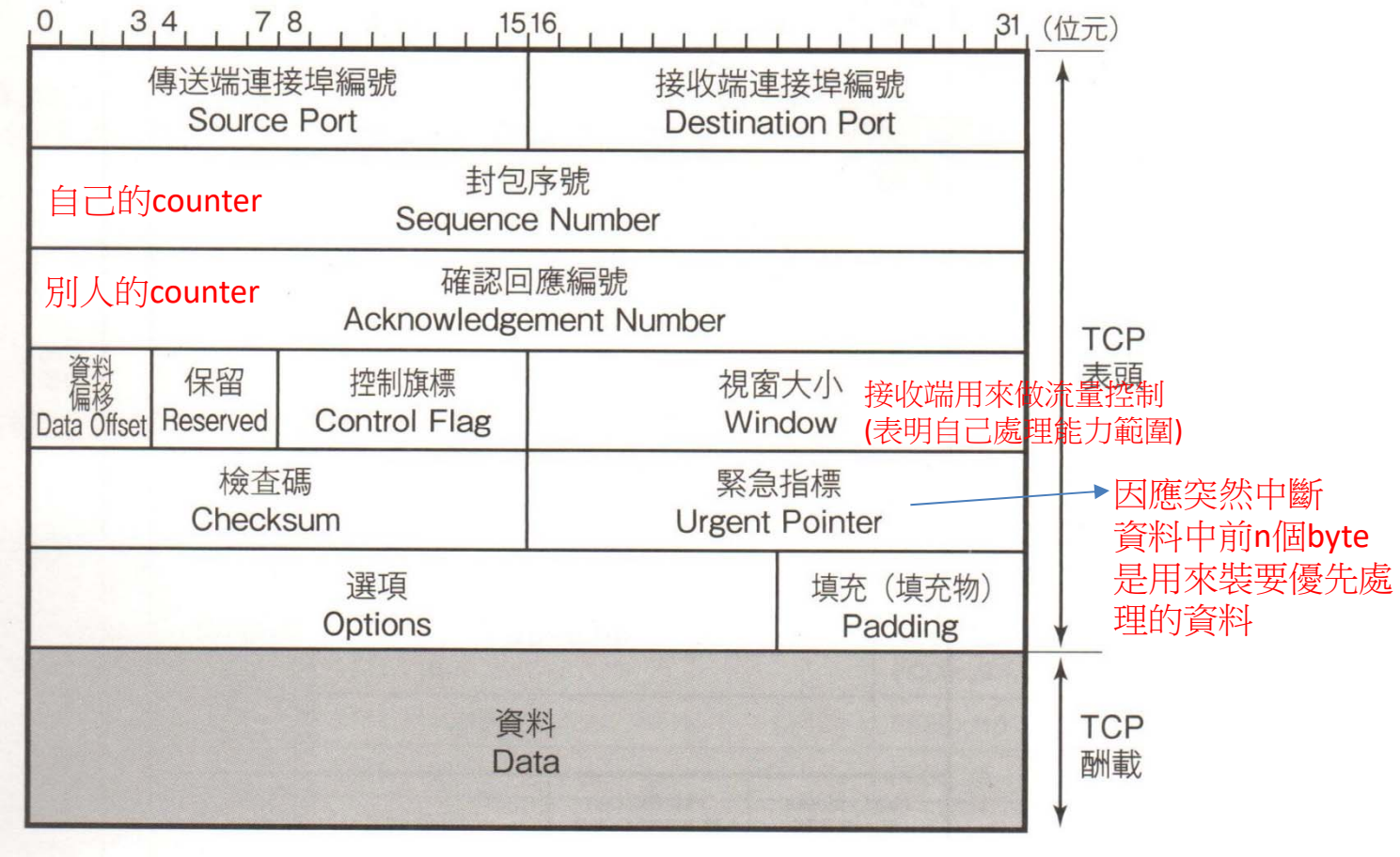
Multicast Receiver

```
let dgram = require('dgram');
let server = dgram.createSocket({type: 'udp4', reuseAddr: true});

server.bind(2390, () => {
    server.addMembership('239.255.255.250');  訂閱群播訊息
});

server.on('message', function (message, remote) {
    …
});
```

收到群播訊息時的處理方式

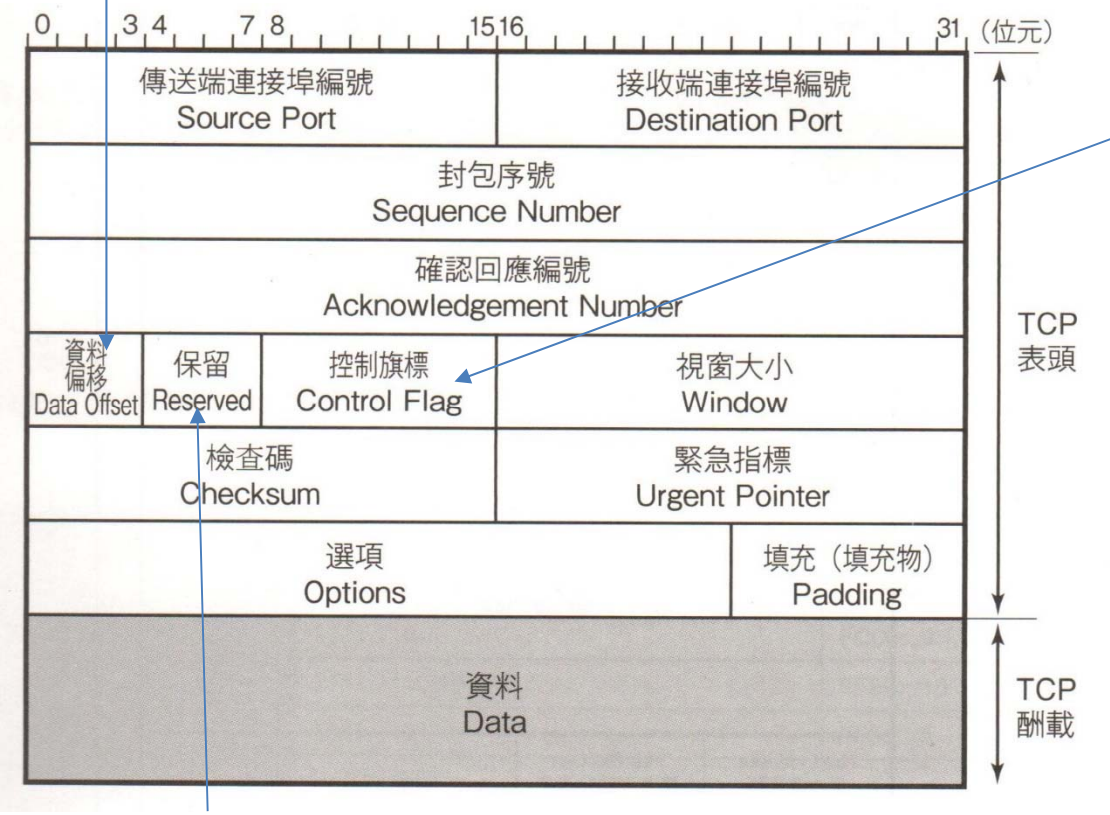
Lab説明

TCP 封包格式


0
3 4
7 8
15 16
31
(位元)
傳送端連接埠編號
Source Port
接收端連接埠編號
Destination Port
自己的counter
封包序號
Sequence Number
別人的counter
確認回應編號
Acknowledgement Number
資料偏移
Data Offset
保留
Reserved
控制旗標
Control Flag
視窗大小
Window
接收端用來做流量控制
(表明自己處理能力範圍)
檢查碼
Checksum
緊急指標
Urgent Pointer
因應突然中斷
資料中前n個byte
是用來裝要優先處
理的資料
選項
Options
填充（填充物）
Padding
TCP
表頭
資料
Data
TCP
酬載

TCP 封包格式

Header length (4)



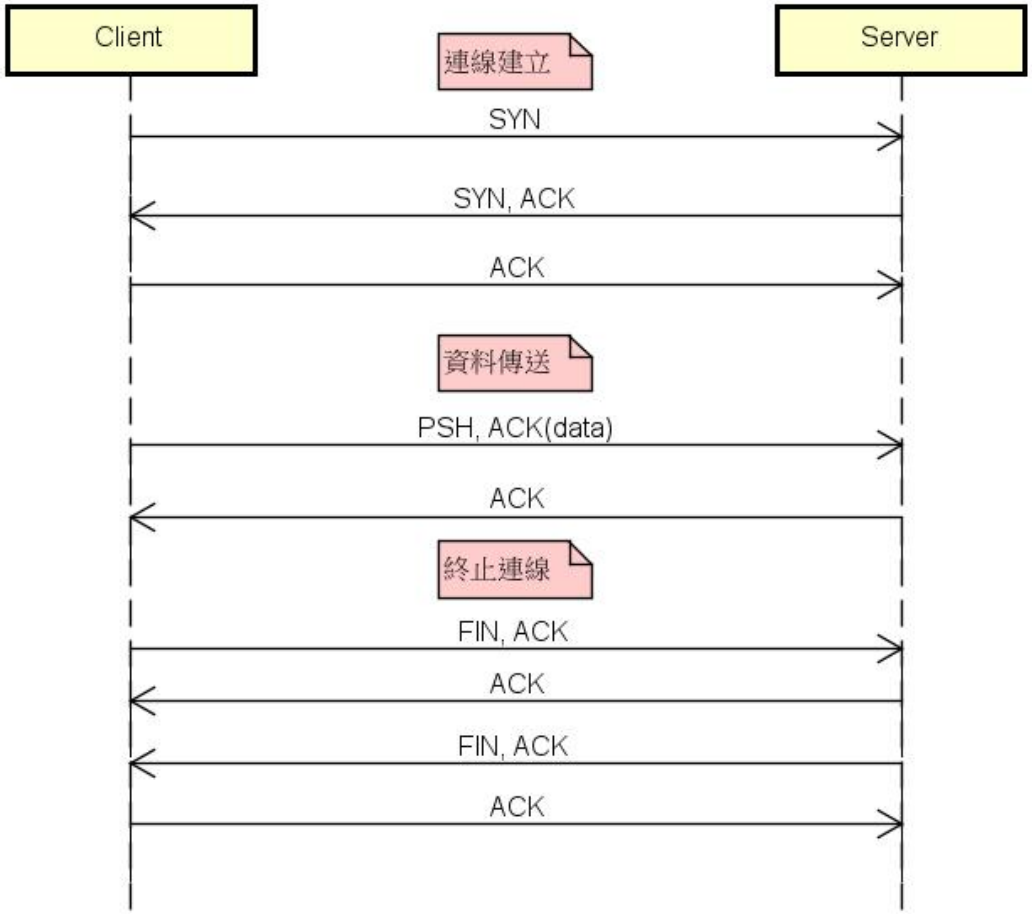
Reserved (6)

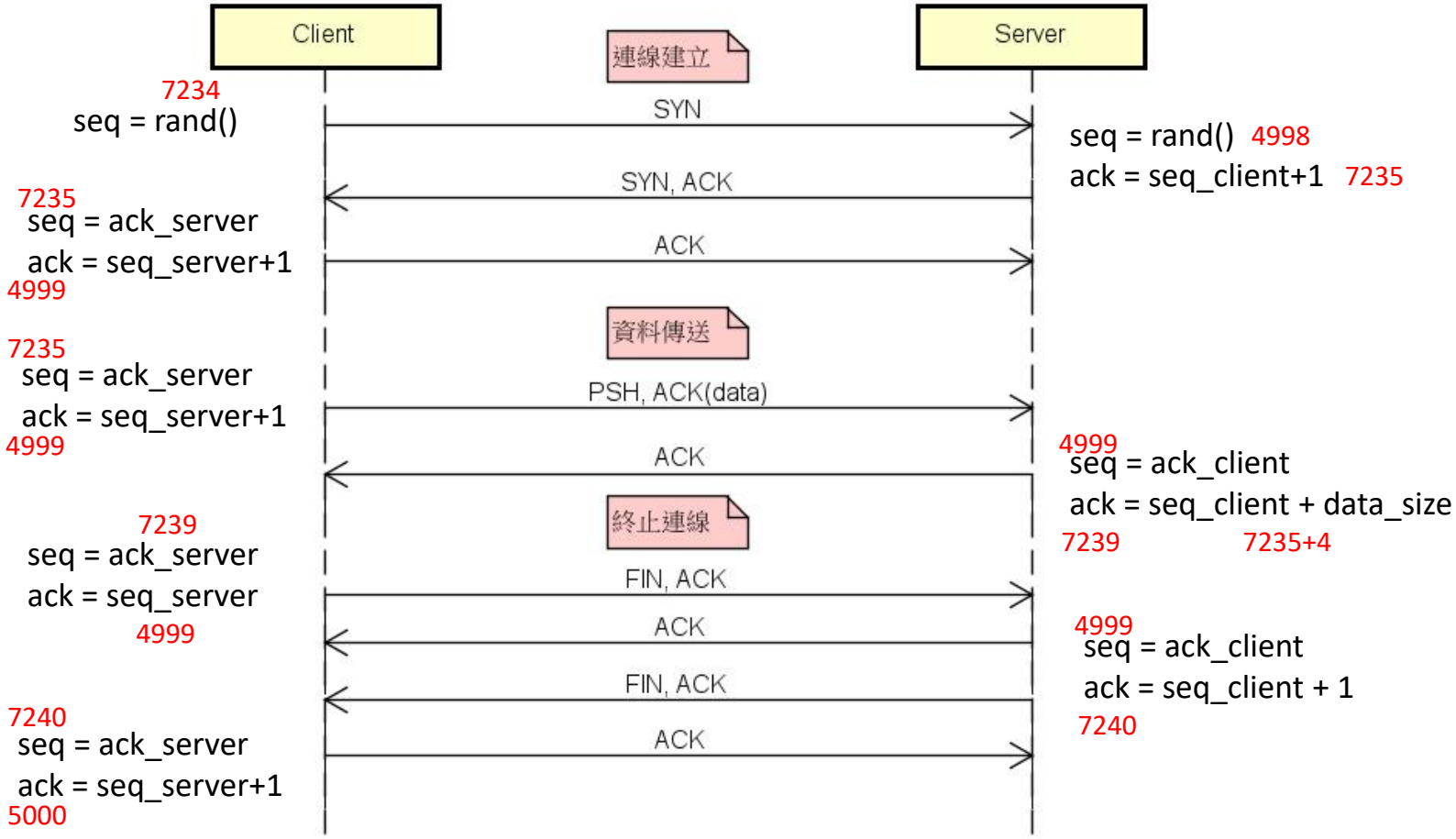
Flags (6) : 控制資料的傳輸與連結

URG : 有使用緊急指標
ACK : 封包是否有ACK
PSH : push function
RST : 重設連結
SYN : 建立順序號碼
FIN : 傳送資料到此為止

例:
SYN, ACK=010010
ACK=010000
PSH, ACK=011000
FIN, ACK=010001

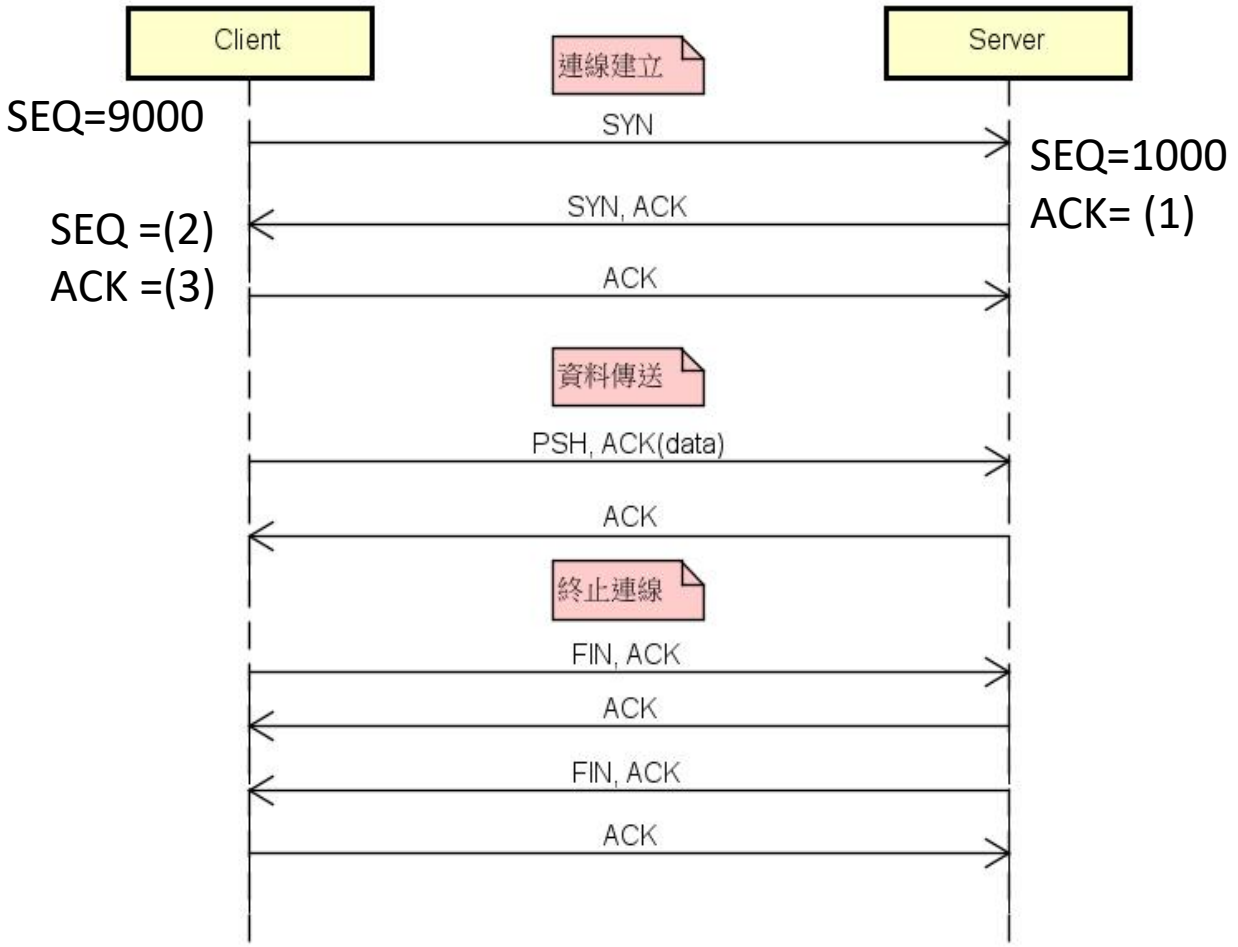
TCP 3-way Handshaking





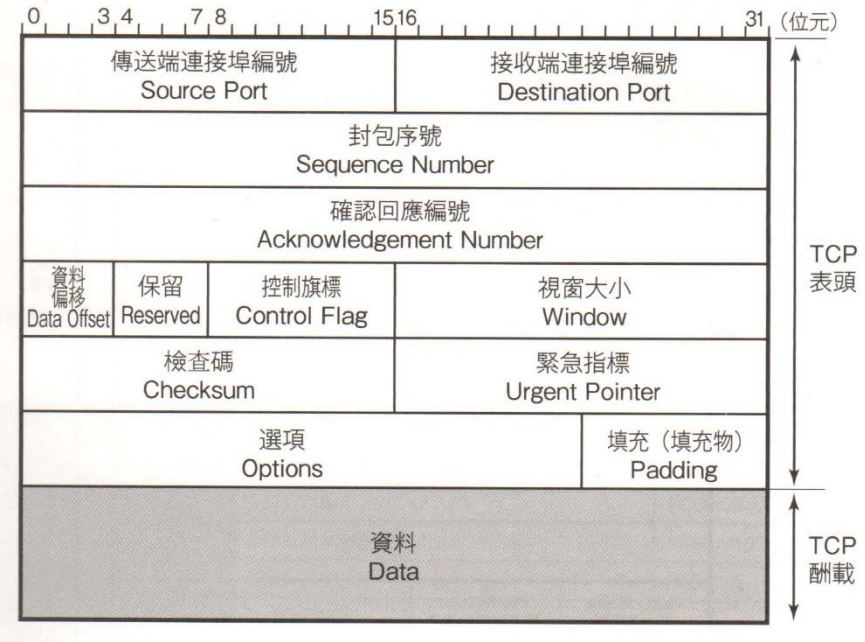
Seq: 自己的counter

ACK: 別人的counter

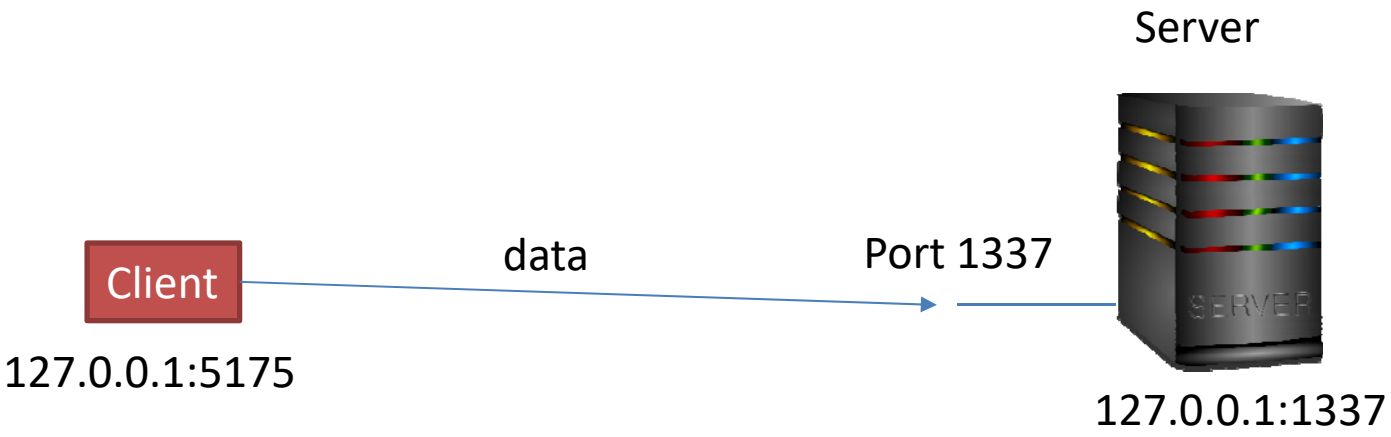

Client
Server
連線建立
SEQ=9000
SYN
SEQ=1000
ACK= (1)
SYN, ACK
SEQ =(2)
ACK =(3)
ACK
資料傳送
PSH, ACK(data)
ACK
終止連線
FIN, ACK
ACK
FIN, ACK
ACK

封包觀察

No.	Time	Source	Destination	Protocol	Length	Info
257	6.206781	127.0.0.1	127.0.0.1	TCP	56	5175 → 1337 [SYN] Seq=0
258	6.206891	127.0.0.1	127.0.0.1	TCP	56	1337 → 5175 [SYN, ACK]
259	6.206913	127.0.0.1	127.0.0.1	TCP	44	5175 → 1337 [ACK] Seq=1
260	6.211008	127.0.0.1	127.0.0.1	TCP	48	5175 → 1337 [PSH, ACK]
261	6.211094	127.0.0.1	127.0.0.1	TCP	44	1337 → 5175 [ACK] Seq=1
262	6.212223	127.0.0.1	127.0.0.1	TCP	44	5175 → 1337 [FIN, ACK]
263	6.212239	127.0.0.1	127.0.0.1	TCP	44	1337 → 5175 [ACK] Seq=1
264	6.215214	127.0.0.1	127.0.0.1	TCP	44	1337 → 5175 [FIN, ACK]
265	6.215235	127.0.0.1	127.0.0.1	TCP	44	5175 → 1337 [ACK] Seq=6



傳送範例


Server
Client
data
Port 1337
SERVER
127.0.0.1:5175
127.0.0.1:1337

TCP Server

- 伺服器

```
let net = require('net');
let server = net.createServer(function(socket) {
    socket.on('data', function(data) {
        ……
    });
});
server.listen(1337);
```

net.Socket

Buffer or String

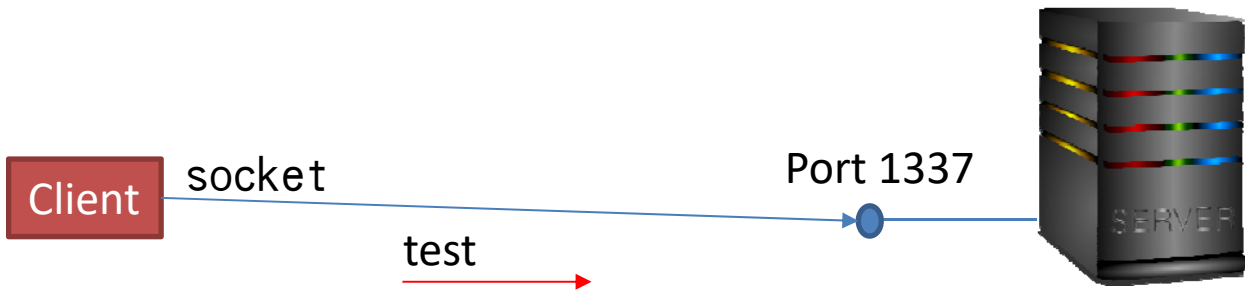
當收到資料時要做什麼?

Port 1337

練習查 API

TCP Client

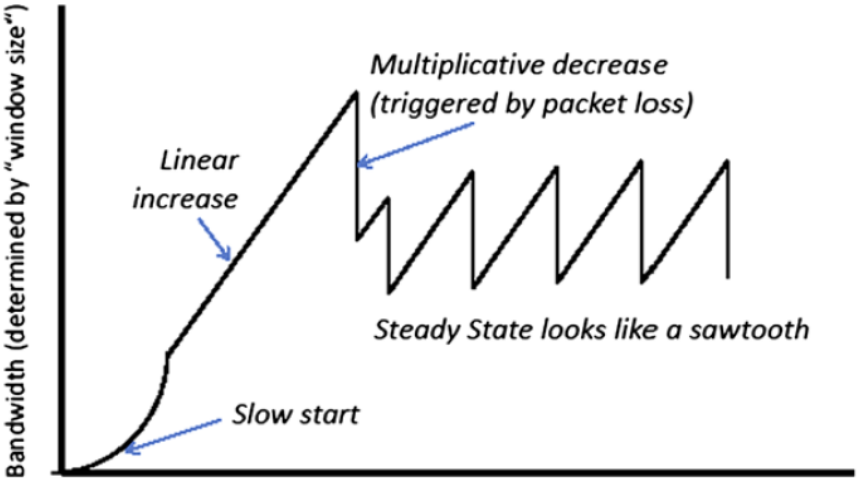
```
let net = require('net');

let client = new net.Socket();

client.connect(1337, '127.0.0.1', function () {
        console.log('connected');
        client.write('test', () => console.log('written'));
        client.end();
    }
);
```



End to End Argument

Saltzer, J. H., Reed, D. P., & Clark, D. D. (1984)

- 意義
  - Not every component needs to guarantee every property
  - the best way to achieve network reliability is not necessarily to make the lowest layers of the network reliable
    - Lower layers : extremely fast and mostly reliable
    - Endpoints: ensure the reliability



Q & A